

COPYRIGHT NOTICE

© Precision Software Limited
6 Park Terrace
Worcester Park
Surrey
England KT4 7JZ
(081) 330 7166

North America:
Precision Inc.
8404 Sterling Street
Irving, TX 75063
USA
(214) 929 4888

First Edition January 1991

This manual, and the Superbase software ('Superbase') described in it, contain confidential information proprietary to Precision Software Limited and are copyrighted with all rights reserved. Neither the whole nor any part of the manual or of Superbase may be copied, reproduced, transmitted, transferred, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written authorization of Precision Software Limited.

Conditions of Authorization of Backup Copy – You may make one backup copy of Superbase, but not of the manual. Every copy must include the same proprietary and copyright notices as the original. Installation of a copy of Superbase on your hard disk is permitted subject to these conditions. You may not make an extra copy of Superbase for the purpose of using it on another computer. You may not sell, loan, or give away a copy of Superbase.

Resale or Disposal – The Software is licensed only to you, the Licensee, and may not be transferred to anyone else without the prior written consent of Precision Software. Any authorized transferee of the Software shall be bound by the terms and conditions of the Licence and Limited Warranty. Upon authorized transfer you shall destroy all copies of the Software not transferred and any written materials not transferred. In no event may you transfer, assign, rent, lease, sell or otherwise dispose of the Software on a temporary or permanent basis except as expressly provided herein.

Trademarks – Superbase, Superplan, Logistix and the VCR Browsing Control design are trademarks of Precision Software Limited.
Windows, MS-DOS and Excel are trademarks of Microsoft Corporation.
IBM-PC is a trademark of International Business Machines Corporation.
Lotus 1-2-3 is a trademark of Lotus Development Corporation.
dBase is a trademark of Ashton-Tate Inc.
PC Paintbrush is a trademark of Zsoft Corporation.
GIF is a trademark of Compuserve Inc.
AmigaDOS, Workbench and Intuition are trademarks of Commodore Amiga Inc.
Xerox is a trademark of Xerox Corporation.
Enable is a trademark of The Software Group.

SUPERBASE AND AMIGA WORKBENCH

To run Superbase under Amiga Workbench, you need to license and install Amiga Workbench Release 1.3 or later.

LIMITED WARRANTY

If there is a physical defect in the Superbase program disk or manual, Precision Software Limited will replace the defective item free of charge, provided you return the item to be replaced with proof of purchase to Precision Software Limited within 90 days of the date of purchase. In some countries the replacement period may be different; check with your software supplier.

The contents of this manual are subject to change without notice and do not represent any commitment on the part of Precision Software Limited.

No warranty is made with respect to Superbase, its quality or performance. Any and all warranties for merchantability and/or fitness for a particular purpose are expressly excluded. The Superbase software is sold 'as is,' and you, the purchaser, are assuming the entire risk as to its quality and performance.

In no event shall Precision Software Limited be liable or responsible to the purchaser or any other person or entity for direct, indirect, special, incidental, or consequential damages, including but not limited to interruption of service and loss of business or anticipatory profits, resulting from any defect in or misunderstanding of the software or its documentation, even if Precision Software Limited has been advised of the possibility of such damages.

The warranty and remedies set out above are exclusive and in lieu of all others, oral or written, express or implied.

Some countries do not allow the exclusion or limitation of implied warranties or liabilities for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

REGISTRATION AND SUPPORT

Please complete and return the Registration Card, which is included in your Superbase package.

Only registered users are eligible for 30 days' free software support from time of registration. If you do not return the card, we will not be able to help you.

Please remember, when requesting support, to make available:

- Your registration number. See your program disk or registration card.
- The version number of your copy of Superbase. This is displayed in the 'About' requester available from the Project menu.
- A clear statement of your problem.

While we are proud of our ability to help users solve their problems, we ask you to remember that only a precise description of a problem can lead to a precise solution.

Before You Begin ...

Read Me First

With this package you will find a set of pages titled IMPORTANT: READ ME FIRST. You should read them before starting to use Superbase. The Read Me First pages include:

- Package Contents check list.
- Instructions for installing Superbase.
- Instructions for running Superbase.

You should have already installed Superbase. If you haven't done so, the procedure is simply a matter of running the Setup program on the Superbase setup disk:

1. Insert the Superbase Setup disk in any floppy drive.
2. Open the disk window by double-clicking on the disk icon.
3. Double-click on the "Setup" icon.
4. Follow the installation instructions as they appear on screen.

README.TXT

The Superbase program disk includes a document called README.TXT. This document describes the changes made to Superbase between the time your Superbase User Guide went to press and the time the final modifications were made to the software. README.TXT is an ASCII text file. You can read it either with the Superbase Text Editor or by double-clicking on its icon.

The Superbase Documentation

The Superbase manuals are designed to allow several ways of becoming familiar with the program.

Volume 1 is a User Guide for the Database and Text Editor, with an introductory tutorial and a reference summary. The User Guide is structured around practical topics, so that you can locate 'how to' instructions easily. The Table of Contents shows the arrangement of topics.

Volume 2 contains a User Guide for the Form Designer and a Keyword Reference Guide for Superbase's Database Management Language (DML).

Volume 2 also includes an Applications Guide, in which a range of worked examples shows how Superbase can be used to build effective database applications.

CONTENTS

Before You Begin ...	iii
Read Me First	iii
README.TXT	iii
The Superbase Documentation	iii
 1 GETTING TO KNOW THE FORM DESIGNER	 1-1
Starting the Form Designer	1-3
The Form Designer Menus	1-3
The Toolbox	1-4
Opening a Form	1-4
Selecting Objects	1-5
Creating a Form	1-6
Where To Go From Here	1-7
 2 THE TOOLBOX	 2-1
Graphic Objects	2-1
Text	2-4
Data and Logical Objects	2-4
Moving Objects	2-5
Sizing Objects	2-6
Color Selector	2-6
Palettes	2-7
Object Editing Tools	2-8
 3 DESIGNING AND EDITING FORMS	 3-1
Editing Aids	3-2
Placing Objects on the Form	3-5
Saving Forms	3-9
Editing Forms	3-9
Managing Pages	3-12
 4 TRANSACTION LINES	 4-1
 5 DEFINING DATA ENTRY ORDER	 5-1
 6 LINKING FILES	 6-1
 7 FORM CALCULATION FORMULAS	 7-1

8 FORM VALIDATIONS	8-1
9 COMMANDS AND CONTROLS	9-1
Pushbuttons	9-1
Commands Without Pushbuttons	9-3
Interfacing to a DML Program	9-3
Check Boxes	9-4
Radio Buttons	9-6
Using Form Controls in DML	9-7
10 GENERATING REPORTS	10-1
Overview	10-1
Example Report Form	10-3
Example Report Program	10-3
Field Selection	10-5
Heading	10-6
Footing	10-7
Sorting by Group	10-7
Subtotals and Other Report Functions	10-8
Global	10-10
Options	10-10
Test	10-11
Saving the Report Form	10-12
Formatting Report Output	10-12
Running a Report	10-12
Modifying a Report Program	10-14
11 PRINTING FORMS	11-1
12 HOUSEKEEPING	12-1
13 COMMAND REFERENCE SUMMARY	13-1
14 INTRODUCTION TO DML	14-1
Using this Guide	14-1
15 DML OVERVIEW	15-1
Operating Modes	15-1
Keywords and Reserved Words	15-1
Commands and Statements	15-2

Variables	15-2
File Types	15-4
Fields	15-4
Date and Time	15-5
Functions	15-6
Operators	15-8
Constants	15-13
Expressions	15-13
Line Format and Labels	15-14
Syntax	15-15
SELECT Commands	15-16
Query Commands	15-17
Form Creation Using DML	15-19
 16 THE PROGRAM EDITOR AND COMMAND LINE	 16-1
Creating a New Program	16-2
Editing a Program	16-4
Cut and Paste	16-5
Using the Command Line	16-5
Loading a Program	16-7
Saving a Program	16-7
Running a Program	16-8
Creating a Start Up Program	16-8
 17 KEYWORD REFERENCE GUIDE	 17-1
Syntax Conventions	17-1
? Commands	17-3
?	17-7
? DIRECTORY	17-8
? LIST	17-9
? MEMORY	17-9
? QUERY	17-10
? STATUS	17-11
? TEXT	17-12
ABS	17-13
ADD	17-14
ADD FORM	17-17
ADD FORM REPLICATE	17-19
AFTER GROUP	17-20

AFTER REPORT	17-21
AFTER SELECT	17-21
ASC	17-22
ASK	17-23
ATN	17-24
BEFORE GROUP	17-25
BEFORE SELECT	17-25
BELL	17-26
BLANK	17-27
BREAK	17-28
CALL	17-29
CHAIN	17-30
CHR\$	17-30
CLEAR	17-31
CLOSE COMMS	17-32
CLOSE FIELDS	17-32
CLOSE FILE	17-33
CLOSE FORM	17-33
CLOSE INPUT/OUTPUT	17-34
CLS	17-35
COL	17-35
COMMS ?	17-36
COMMS FILE	17-36
COMMS FILE GET	17-37
COMMS GET	17-38
COMMS INPUT	17-38
COPY	17-39
COS	17-40
COUNT	17-40
CREATE	17-41
CREATE FORM	17-42
CREATE INDEX	17-44
CREATE PAGE	17-44
DATA	17-45
DATE\$	17-46
DATEBASE	17-47
DAY	17-48
DAY\$	17-49

DAYS	17-50
DEBUG	17-51
DELETE	17-51
DIM	17-52
DIRECTORY	17-53
DISKSPACE	17-53
DISPLAY	17-54
EDIT	17-55
EJECT	17-56
END	17-56
END FOOTING	17-57
END GROUP	17-57
END HEADING	17-58
END REPORT	17-58
END SELECT	17-59
ENTER	17-59
EOF	17-62
ERASE	17-63
ERR\$	17-64
ERRNO	17-64
EXECUTE	17-65
EXISTS	17-66
EXP	17-67
EXPORT	17-68
FCASE\$	17-71
FILE	17-71
FIX	17-72
FN alpha	17-74
FN ansi	17-74
FN dec	17-75
FN ext	17-75
FN fact	17-76
FN fv	17-76
FN hex	17-77
FN ibm	17-78
FN name	17-78
FN nper	17-79
FN numeric	17-79

FN path	17-80
FN pmt	17-80
FN pv	17-81
FN rate	17-82
FN root	17-82
FN sln	17-83
FN sys	17-84
FOOTING	17-85
FOR TO NEXT	17-85
FORM	17-87
FORMAT\$	17-88
FOUND	17-89
FREE	17-90
GET	17-90
GOSUB	17-91
GOTO	17-92
GROUP	17-93
HEADING	17-94
HOME	17-94
HRS	17-95
IF THEN ELSE	17-96
IMPORT	17-99
INDEX	17-102
INPUT	17-103
INSTR	17-104
INT	17-105
KEY	17-105
LABELS	17-107
LCASE\$	17-108
LEFT\$	17-109
LEN	17-110
LET	17-110
LIST	17-112
LOAD	17-112
LOCATE	17-113
LOG	17-114
LOOKUP	17-115
LTRIM\$	17-116

MACRO	17-117
MAKE	17-117
MAX	17-118
MEAN	17-118
MENU	17-119
MENU CLEAR	17-121
MENU ON/OFF	17-121
MERGE	17-122
MID\$	17-123
MIN	17-124
MINS	17-124
MOD	17-125
MODIFY	17-126
MONTH	17-126
MONTH\$	17-127
MOUSE	17-128
MOUSE ON/OFF	17-129
NEW	17-130
NEWLINE	17-130
NOW	17-131
NUMBASE	17-132
ON ERROR	17-133
ON GOSUB	17-134
ON GOTO	17-135
OPEN	17-135
OPEN COMMS	17-137
OPEN FIELDS	17-138
OPEN FILE	17-139
OPEN FORM	17-140
OPEN INDEX	17-141
ORDER	17-141
OUTPUT TO	17-145
PAD\$	17-146
PANEL	17-147
PASSWORD	17-147
PCOL	17-148
POSITION	17-149
PRINT	17-151

PRINTER	17-153
PROTECT	17-153
PROW	17-154
QUIT	17-154
READ	17-155
RECCOUNT	17-155
REM	17-156
REMOVE FILE	17-157
REMOVE FROM	17-158
REMOVE INDEX	17-158
RENAME	17-159
REORGANIZE	17-159
REPLICATE	17-160
REPORT	17-161
REQUEST	17-162
RESTORE	17-167
RESUME	17-168
RETURN	17-169
RIGHT\$	17-170
RND	17-170
ROW	17-171
RUN	17-172
SAVE	17-173
SAVE FILE	17-173
SAY	17-174
SD	17-174
SECS	17-175
SELECT	17-176
SELECT CASE	17-178
SELECT CURRENT	17-180
SELECT DUPLICATE	17-180
SELECT FIRST	17-181
SELECT FORM NEXT/PREVIOUS PAGE	17-182
SELECT FORM ROW	17-183
SELECT KEY	17-184
SELECT LAST	17-185
SELECT NEXT	17-185
SELECT PREVIOUS	17-186

SELECT REMOVE	17-186
SELECT WHERE	17-187
SER	17-189
SET	17-190
SET BUFFERS	17-191
SET EDIT	17-191
SET EDIT ATTR	17-192
SET FIELDS	17-192
SET FILE MAX	17-193
SET FN KEY	17-193
SET FORM	17-194
SET FORM ATTR	17-194
SET FORM BF	17-195
SET FORM BG	17-196
SET FORM FG	17-196
SET FORM IT	17-197
SET FORM PAGE	17-198
SET FORM TEXT	17-198
SET FORM UL	17-199
SET FORM USING	17-199
SET HEADING	17-200
SET NOW	17-200
SET PAGE	17-201
SET PAGING	17-201
SET PG	17-201
SET PG REQUEST	17-202
SET POSITION	17-203
SET PRINTER	17-203
SET PRINTER REQUEST	17-204
SET RECORD	17-204
SET REQUEST	17-204
SET REQUEST HEADING	17-205
SET STATUS	17-205
SET TABLE	17-206
SET TEXT	17-206
SET TODAY	17-206
SET View or Form	17-207
SGN	17-208

SHOW	17-208
SIN	17-209
SPACE\$	17-210
SQR	17-211
STORE	17-211
STR\$	17-213
SUM	17-214
TAN	17-215
THOUSECS	17-215
TIMES\$	17-216
TIMEVAL	17-216
TODAY	17-217
TRIMS\$	17-217
UCASE\$	17-218
UPDATE	17-219
UPDATE FORM ROW	17-220
VAL	17-221
VAR	17-222
VIEW	17-222
WAIT	17-223
WAIT COMMS	17-225
WHERE	17-225
WHILE WEND	17-226
YEAR	17-227
18 DML SUMMARY	18-1
19 USING AREXX WITH SUPERBASE	19-1
Communicating with Applications from Superbase	19-1
Controlling Superbase from AREXX	19-3
INDEX	I-1

1 GETTING TO KNOW THE FORM DESIGNER

The Form Designer is a design system for producing input and browsing screens, forms for printing, and reports. The user combines graphic, data, and logical objects together using a range of editing tools and techniques. The resulting form may then be opened from within the database.

The facilities in the Form Designer are designed to match its role as a complementary program to Superbase itself. Its primary purpose is to generate forms for use in database applications.

Forms provide the highest level of database design surface in the Superbase system, allowing users to construct complete applications with little or no need for programming knowledge. Where necessary, Superbase's detailed procedural Database Management Language offers a path for the development of full-scale database applications in both single and multi-user environments.

Forms usually consist of a background of graphic objects such as areas and lines, onto which fields from one or more existing database files may be placed. You may also include calculation formulas and commands on a form.

When you enter data into a form which includes fields from more than one file, Superbase automatically creates the correct set of records for all the files. This is also possible where one file holds multiple records in relation to a single record in another file.

Report forms are created with the special Report Generator menu, which also creates an editable Superbase DML program.

Pages are the building blocks of forms. You may create multi-page forms, and save the individual pages separately as a library of designs.

Designing Screens

Screen design is always geared to a specific part of a specific application. The Form Designer allows you to create any number of forms for use with a single file or set of files. The files with which a form is to be used must be set up within the database before you can create the form.

In a typical application, you will require at least one form for data entry. If more than one person will be entering or editing data, you may prefer to create separate forms for each person. Each form can show just a selection of the fields from a file, so you can build in protection for your data by excluding sensitive fields where necessary.

In addition, you may designate any field on a form as either Read Only or Display Only, thus preventing it from being changed or printed out, as appropriate. Files in the database may have up to three levels of password protection.

Many applications include a 'one-to-many' relationship between two files. The Form Designer allows you to define an area of a form as a block of repeating fields. Each group of fields represents a 'transaction' – one record in the file on the 'many' end of the one-to-many relationship.

Both in data entry and in browsing, such a form provides the ability to manage sets of related records from multiple files without the need for programming.

Browsing forms may well be different from data entry forms. They may be designed for on-line querying of a database, with the option to produce an instant printout. 'Relational browsing' is a term sometimes used to describe browsing with a multi-file form. You may step through or look up the records in any of the files involved in the form, and Superbase will retrieve all related records.

Designing Forms for Printing

The Form Designer achieves a high degree of page fidelity. You can choose the physical size of your page from the sizes available from your printer driver.

The arrangement of objects on the form may use absolute physical measures of inches or centimeters, derived from the Ruler and Grid feature. One inch on the ruler will normally equal one inch on the printed page.

This degree of fidelity allows you to design a full range of office stationery with the Form Designer. If you wish, a screen form may be the same as the form that is printed. The inexperienced user may be assigned to enter data from paper forms that are identical with the screen. Likewise, data to be distributed may be viewed in the same format in which it is actually printed.

Designing Reports

The Report Generator menu in the Form Designer includes commands for creating a special type of form. They allow you to lay out a report on the screen using the easy point-and-click techniques and editing tools of the Form Designer.

Reports may include multi-line titles and headings, multi-level sorting, and fields from different files. Specialized functions for report analysis are available.

When you save a report form, the Form Designer generates a DML program. You may then run this report program directly from within Superbase, or if you prefer edit it in order to add extra features.

Starting the Form Designer

The Form Designer program is called SBFD4. You may start it from Workbench like any other Workbench application.

- Forms should be stored in the same directory as the files they use. You will usually change to this directory when you open a database file from which to select fields.

Relationship to Superbase

You may also run the Form Designer from within Superbase.

1. Select Modify Form from the Project menu.
 - ☐ If a form is already open in Superbase, the Form Designer opens it ready for editing.

When you reactivate Superbase, it checks to see whether the form it has in memory has changed on disk. If it has, Superbase reloads the form.

This allows you to save the form you have been editing, and simply return to Superbase to use the new version of the form.

The Form Designer Menus

Project	
New ..	AN
Open >>	
Save	AS
Save as ..	AA
Remove ..	
SB File Close ..	
Directory list ..	AL
Status ..	AZ
Printer Setup ..	
About ..	
Quit	AQ

Edit	
Undo	AU
Cut	AX
Copy	AC
Paste	AV
Clear	A-
Tools	
Snap to Grid	AY
Ruler/Grid ..	AD
Crosshairs	AH
Auto Field names	
Reduced	AR

Page	
New	
Open	
Save as ..	
Remove	
Setup	
Hierarchy ..	
Clear	
Erase ..	
Next	A>
Previous	A<
Go to ..	AG

Define	
Font ..	AF
SB File Link	AK
Data Entry Order ..	AE
Transaction Lines ..	AT
Color palette ..	
Resolution >>	

Report	
Heading	
Select	
Footing	
Group ..	
Global	
Options ..	
Test	

Together with the Toolbox described in the next section, the Form Designer menus contain all the commands for controlling the system. This User Guide assumes you are familiar with the Superbase system, and hence Workbench itself, so we do not repeat here explanations of how to use menus, requesters, and other controls. These topics are all covered either in Volume 1 or in your Amiga documentation.

Project Menu. The Project menu commands allow you to open, close, remove, and save forms. Two commands give access to Superbase database file definitions, and a group of housekeeping commands provides for status reporting, form printing and directory listing.

Edit Menu. This menu complements the familiar Cut and Paste commands with a set of editing aids, including Snap to Grid and Crosshairs.

Page Menu. The Page menu contains the commands for managing pages, which are the building blocks for forms.

Define Menu. A set of commands for selecting text font, setting up links between database files, specifying transaction blocks and defining data entry order.

Report Menu. The special commands for creating a report form and generating the code for a Superbase report program to be executed in the database.

The Toolbox

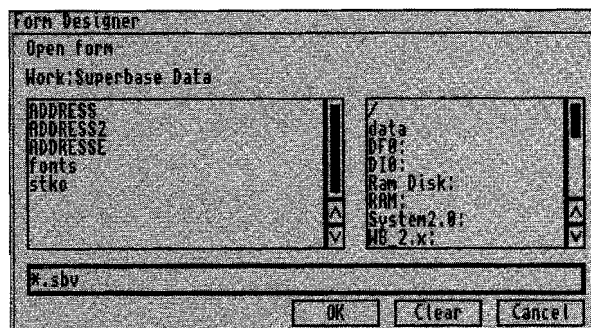
The Form Designer toolbox is the most important resource for the application designer. You select from a range of graphic, data, and logical object types, then click to place an object on the form. Objects may have many different attributes, from Read Only status to pattern and color. Tools are provided for moving and sizing objects.

See Chapter 2 The Toolbox for more details.

Opening a Form

Some model forms have been supplied with your example data. To see what a form looks like, open one now.

1. Select the Project Open Form command.
2. When the requester appears, change to the **Model** directory, or to the directory in which you have placed the example files.
3. In the left-hand list, click the file **Clients**. The Form Designer opens and displays the form.



You may now use the Form Designer's full range of tools to experiment with and modify this form. If you wish you may save your experiments under different names and use them in Superbase.

Selecting Objects

An important principle in the Form Designer is that an object should always be selected before you may operate on it. The converse is also true, that any operation works on the selected object. You should understand selection techniques at an early stage.

Selecting a Single Object

Point to the object you wish to select. Double-click the mouse button.

The Form Designer outlines the selected object with a line of dashes.

Note

Distinguish the dashed selection line from the dotted line that surrounds fields.

Selecting a Group of Contiguous Objects

Hold down the Control key and click the mouse button. Hold it down, and drag a box around the group of objects you wish to select. Release the mouse button and the CTRL key.

The Form Designer outlines all the selected objects with dashed lines.

Selecting a Group of Discrete Objects

Holding down the SHIFT key, click the objects you wish to select. As you click each one, the Form Designer outlines it with dashes.

Deselecting an Object

You may deselect an object in two ways:

- Press SHIFT while you click on the selected object.
- Select another object.

Selecting an Object for Moving or Sizing

Moving and Sizing are such frequent operations that the selection process has been simplified.

To move a single object:

1. Click the four-way arrow tool in the toolbox.
2. Point to the object you want to move.
3. Click the left-hand mouse button and hold it down.
4. Drag the object to its new location.
5. Release the mouse button.

To size a single object:

1. Click the sizing tool in the toolbox.
2. Point to the object you want to size.
3. Click the left-hand mouse button and hold it down.
4. Drag the object to its new size.
5. Release the mouse button.

Creating a Form

For users who feel confident enough to begin creating forms at this point, the following summary procedure provides a shortcut into the functionality of the Form Designer.

1. Select Project New to clear the work area if necessary.
2. Select the Open SB File command on the Project menu.
3. Select the Superbase database file for which you wish to create a form.
 - ☐ Restrict yourself to a single file to begin with.
4. Select Auto Field Names and Snap to Grid on the Edit menu.
5. Define a grid with the Edit Ruler/Grid command.
 - ☐ Turn on the Characters check box.
6. Click the Field object type (FLD) in the toolbox.

7. Click where you want to place a field.
 - Leave enough room for the field name on the left.
8. Select a field from the list of fields, and click OK.
 - Repeat steps 7 and 8 for each field.
9. Click the round cornered Area object type on the toolbox.
10. Choose a color by clicking the center of the color selector in the middle of the toolbox, then clicking a color from the color palette.
11. Click above and to the left of the top field on the form, drag the outline to surround all the fields, and release.
12. Save the form with the Project Save command.

You may now exit from the Form Designer and open the form from within Superbase. The form will automatically open any files it requires.

Where To Go From Here

The rest of this User Guide covers all the tasks the forms designer must become familiar with. The next two chapters cover the Toolbox and the basic procedures for designing and editing forms.

The following six chapters go through the data related and logical components of forms design. Chapter 10 Generating Reports explains how to design report forms.

A command summary is provided at the end of the guide.

2 THE TOOLBOX

The Form Designer toolbox provides easy and continuous access to the object types and editing choices needed by the application designer.



The left-hand part of the toolbox shows the graphical, data and logical object types. Next to them are the tools for moving and sizing objects.

In the center of the toolbox are the color selection tools and a group of tools for selecting pen and paper, rounded corners and borders. Next to these are tools for display only, text style, read only, currency calculation type, image scaling, and field justification.

The right-hand part of the toolbox includes palettes for color, area pattern, and line pattern.

The default number of colors available under Workbench 1.3 is four. More colors are obtainable by changing the resolution using the Define Resolution command.

Hiding The Toolbox

If you wish to remove the toolbox from the window, select the Tools command from the Edit menu.

Reselect the command to redisplay the toolbox.

Graphic Objects

The graphic objects available to the form designer are appropriate to the function of the program: the creation of forms for database applications. The range is restricted to rectangular objects, but images may be imported from design software packages if you need circles and polygons on your forms.

Boxes



A box is a rectangular area on a form. It may exceed the window size, and be drawn in any color or line thickness available. A bordered area tool type is also available.

Use boxes to group fields in logical arrangements, or to contain text objects.

1. Select the box object type from the toolbox.
2. Select a pen color and a line pattern.
3. Select the rounded corners tool if required.
4. Click where you want the top left-hand corner of the box to appear.
 - ☐ You may wish to use a grid or the Snap to Grid or Crosshairs editing tools to help you position the box accurately.
5. Drag the box outline to the required extent, then release the mouse button. The Form Designer draws the box in the specified color and line pattern.

To edit a box, first select it in the normal way. Then you may size it, or change its color or line pattern.

To size the box, select the Size tool, point at the box and drag it to the required size.

Lines



Lines are useful for dividing boxes or areas of the screen either horizontally or vertically.

1. Choose the line object type from the toolbox.
2. Choose your required line pattern from the toolbox.
3. Specify a color.
4. Click where you want to anchor one end of the line.
 - ☐ You may wish to use a grid or the Snap to Grid or Crosshairs editing tools to help you position the line accurately.
5. Drag the line to the required extent, then release the mouse button.

The Form Designer draws the line in the specified color and pattern.

To edit a line, first select it in the normal way. Then you may size it, or change its color or pattern.

To size the line, select the Size tool, point at the line and drag it to the required length.

Areas



An area is a rectangle on a form. It may exceed the window size, and be drawn in any color or pattern available. A bordered area tool type is also available. The border color of a bordered area is set separately from foreground or background colors in the toolbox.

Use areas to identify parts of a form by color coding them, or simply to enhance the esthetic appeal and usability of the screen.

1. Select the square or round cornered area object type from the toolbox.
2. Choose your required area pattern from the toolbox.
3. Specify a color.
4. Select the rounded corners tool if required.
5. Click where you want the top left-hand corner of the area to appear.
 - ☐ You may wish to use a grid or the Snap to Grid or Crosshairs editing tools to help you position the area accurately.
6. Drag the area outline to the required extent, then release the mouse button.

The Form Designer draws the area in the specified color and pattern.

To edit an area, first select it in the normal way. Then you may change its color or pattern.

To size the area, select the Size tool, point at the area and drag it to the required size.

Images



You may wish to place images on a form to improve the design or to allow the printing of forms with logos or special lettering. Small images such as arrows or symbols may help to make your form easier to use.

1. Select the image object type from the toolbox.
2. Click where you want the image to appear.
3. Drag the expandable box to the required size.
4. Select an image from the list.
 - ☐ Remember that changing directory to load an image does not return you to where you started.
5. The image is positioned at the specified point.
 - ☐ If the image scaling tool is selected in the toolbox, the Form Designer adjusts the image accordingly. Windows metafiles (WMF) are always scaled.

The following image file types are supported:

PCX GIF Some IFF

Variations in image generating software make it impossible to guarantee that all such images can be read.

Text



You may wish to place text on a form for various reasons: to act as a prompt to the operator, to clearly identify the nature of the application, or to mark out groups of fields more clearly.

Define the characteristics of the text before you start typing it.

Selecting Font

Text and field data may be rendered in any font available to the system. Use the Define Font command to select a font and point size.

- If the font used by a form object is not available when the form is edited or used in the database – for example if a form is created on one workstation and used on another – Superbase and the Form Designer will substitute a similar font. However, the visual appearance of the substitute font may differ substantially from the original.
1. Choose the text style, plain, bold, underline, or italic, from the toolbox.
 2. Select a color for the text, for both pen and paper if necessary.
 3. Select the Pen and Paper tool if you want text to be rendered with a background.
 4. Choose the Text object type from the toolbox.
 5. Click where the text is to appear.
 - ☐ You may want to use a grid or the Snap to Grid or Crosshairs commands to help you position the text accurately.
 6. Type the text, pressing ENTER to finish.

Changing Text Attributes

First, select the Text tool type. Select the text to be edited as you would any object. You may select multiple text objects if you wish.

Now you may vary the color and other characteristics of the text simply by clicking the appropriate tools in the toolbox.

- You may vary the font used for existing objects. Select the object or objects to be changed, then choose the new font.

Data and Logical Objects

The toolbox includes seven data and logical object types. These are represented by five abbreviations and two icons:

FLD	Field		
CLC	Calculation	Check box	
VAL	Validation		
CMD	Command	Radio button	
FNC	Report Function		



These object types are only enabled when the circumstances are appropriate. For example, Field will not be available unless a Superbase file has been opened, and Calculation requires at least one field to have been placed on a form.

All of these objects are dealt with elsewhere in the User Guide, but there are some general points specially relevant to using the toolbox.

- Fields may be rendered in any font and text style.
- Fields and Calculations may be designated as Read Only, Display Only, and justified left, right or center.
- When placing fields and calculations on a form, you may request that field names and borders are supplied automatically. Use the Auto Field Names command and the Borders tool respectively.
- Validations are attached to fields, and are not manipulable as separate objects. Calculations may only have validations attached to them in the File Definition. See Volume 1.
- Check boxes and radio buttons may be designated as Read Only or Display Only, and may be rendered in any colors.

Moving Objects



You will need to move objects on a form while you are creating it.

Select the four-way pointer tool in the toolbox. Now you can click an object and drag it to a new position on the form in one movement.

Alternatively, select the object or objects you wish to move, then select the move tool, and drag the selected objects to their new positions. A group of selected objects moves as one.

- See Chapter 1 for further discussion of selection techniques.

Sizing Objects

Objects on forms frequently need to be resized. This tool allows you to vary the size of any type of object. It is essential to resize long text fields and external file fields so that data can be shown properly.



Select the sizing tool in the toolbox.

Now you can click an object and drag it to a new size in one movement. Only one object may be resized at once. Clicking on an object to size it is always assumed to define the position of the bottom right-hand corner of the object.

- You cannot size a text object. Text objects must be edited. See Chapter 3 Designing and Editing Forms.

Color Selector



Three colors are selected at any one time, for pen, paper, and borders.

The Color Selector in the center of the toolbox shows the current colors in which objects and text will be rendered. There are two areas in the main Color Selector, a center and a surround. These represent the pen and paper colors respectively.

To change the pen color, click the center of the Color Selector, then click the required color.

The pen color of all selected objects changes when a new color is clicked.

To change the paper color, click the surround of the Color Selector, then the required color.

Border Color Selector

Next to the main Color Selector is the Border Color Selector. The color shown here determines the color in which the borders of areas and fields are rendered.

Pen and Paper



When a text object, a field, a calculation, or a command is created, it has both a pen color, in which the characters are drawn, and a paper color, which is used for the background to the character. Areas which are drawn in a pattern of dots also have a foreground color and a background color.

Other types of object are shown with pen color only.

The Pen and Paper tool must be selected if you want the paper color to be drawn when you create an object.

Palettes

There are three palettes in the toolbox: Colors, Areas, and Lines.

Colors are selected with the color selector described above, which also shows the current colors for pen, paper, and borders.

Color Palette

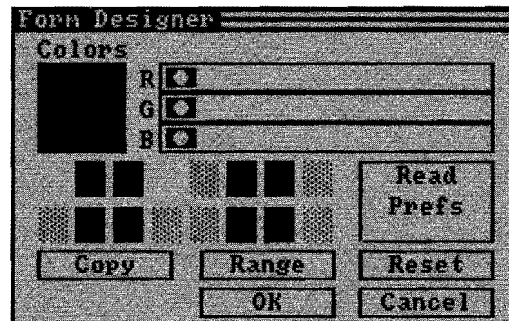
The color palette usually shows four true colors and twelve patterned colors, which are enabled only if your resolution allows more than four colors..



The arrows may be used to display other available colors. Whether these are true or dithered colors depends on the screen driver you have installed.

Redefining the Color Palette

The Color Palette command on the Define menu allows you to redefine the colors used in the toolbox.



Select the command, then click the color you wish to modify. Drag the sliders until you have created the color you want, then click OK.

The new color replaces the previous color in the toolbox display. All objects that used the previous color are redrawn in the new color.

Area Pattern Palette

You can vary the appearance and the color of an area by drawing it in a pattern of dots. The default pattern is solid.



Choose a pattern by clicking in the palette of pattern types. When you create an area it will be rendered in the selected pattern.

If the pen and paper tool is selected, patterned areas will be drawn in two colors.

Line Pattern Palette

Different types of line thickness may be used to differentiate different kinds of box, area, and line.



Choose a line pattern in the toolbox by clicking it in the line pattern palette. If any lines or boxes are already selected, they will be changed to the new pattern.

Object Editing Tools

The toolbox includes a range of special tools for setting and modifying the attributes of objects. Some of the tools may be used with any object type, some with just one.



Field and Area Borders

Field borders are not normally shown when a field is placed on a form. However, if the borders tool is selected in the toolbox, fields, calculations, and commands will be shown with a border drawn in the current pen color.

The line used for field borders is drawn in the finest line pattern.

Areas may also be drawn with borders if this tool is selected. Area borders are drawn in the currently selected line pattern.

Field Justification



Fields may be justified left, center, or right. Click the selector to the required position.

All selected fields and all newly placed fields will be justified as indicated.

Text Style

Text objects and field data may appear in plain, boldface, italic or underlined text styles.



Select the required text style or styles in the toolbox, then create the text object or place a field on the form.



Currency Calculation

The dollar tool is used to designate form calculations as currency type.

When you create a numeric calculation with this tool selected, its results will be shown to two decimal places, right aligned.

If the calculation length is less than four characters, it is formatted as an integer.

Read Only Object

You may wish to protect certain fields from being edited by users.



This tool allows you to designate fields, calculations, and commands as read only, which means that Superbase will not allow you to edit them when using the form for data entry.

You could have different versions of the same form with different fields designated read only according to the security clearance of the user.

To create an object as read only, select this tool and then proceed.

To mark existing objects as read only, first select them, then click the read only tool.

Display Only Object



You can designate any object on the form as display only. This means that it will not be printed when you print forms and data from the database.

This feature allows you to include user prompts or intermediate calculation fields for use in data entry, and be able to use the same form for printing without the unwanted objects.

To create an object as display only, select the monitor icon tool and then proceed.

To mark existing objects as display only, first select them, then click the monitor icon tool.

Image Scale

Form images are those created with the Image object type.



External file images are data files which Superbase can read and show in a external file field box on a form. Both types of image can be assigned the Image Scale attribute.

Form images are used as part of the design of the form. If the Image Scale feature is not selected, an image that is larger than the box you create for it will be clipped to fit. If Image Scale is selected, the image will be scaled to fit the box.

If Image Scale is selected as a characteristic of an External File Field on a form, then external images will be scaled when they are displayed on the form. If the feature is not selected, then external images will be clipped.

3 DESIGNING AND EDITING FORMS

Designing forms requires a mix of creative and logical skills. To be successful, a form must achieve its purpose, which is to provide an interface to a Superbase database. The form designer needs to understand, at a minimum, how to set up a Superbase file definition and how users enter and review data.

A basic form needs only to show some or all of the fields from a database file. No graphical objects or text are needed. However, a form with no text would leave the user unable to distinguish one field from another. So the minimum requirement for a usable form is to complement the fields from the database with their names.

Beyond these minimum requirements, a form should employ graphic features to improve its usefulness. Color, patterns, boxes and lines all contribute to the visual structure of the form, making it easier to understand.

In this chapter, we look at the editing aids available to the form designer, and see how to combine graphic and data elements to produce effective screen forms.

Basic Design Concepts

A form is a picture. Everything on a form should be harmonized to make it a pleasure to work with. The more attention paid to this ergonomic and esthetic aspect of form design, the better users will interact with the software. They will make fewer mistakes, enjoy their work more, and contribute more to your organization.

Perhaps the most basic of concepts for form design is that of focus. The average user scans a page or a screen from left to right and top to bottom, pausing at areas which draw attention to themselves. As a designer, you should take advantage of this.

The fields in most data files can be divided into groups. For example, a simple address file has two groups: the name and the address. A more complex customer record is likely to include financial information. A record for a stock item is still more complex, with fields for identification and description, supplier, prices, and quantities. It makes sense to group these fields together on a form, placing them within separate boxes, or at least separating each group from the others with lines.

The actual positioning of groups on the form depends on the number and size of the groups. A simple rule of thumb is to place the most frequently read data near the center of the screen, or within vertical or horizontal zones that intersect as a broad cross. Leave the corners of the screen for less frequently read data. Remember that you may have multiple pages in a form. Sensitive or very infrequently read data may be placed on a second page. See **Managing Pages** at the end of this chapter.

Text is essential. There should be neither too much nor too little. Since you do not have to use fieldnames, you should omit them where possible. For example, a group of address fields needs only the cue 'Address' next to or above it for the user to see at a glance what the fields are for. On the other hand, you can add extra text to make fields intelligible where the field name is itself enigmatic. **Q1** could be expanded to 'First Quarter,' for example.

Always identify the application for the user. Reserve a heading area for a description of the application and if necessary the specific function. For example, 'ADDRESS FILE: Add New Records' explains what is going on in five words. You could underline or embolden the text, or draw it in contrasting pen and paper colors. Likewise, identify which page of a multi-page form the user is looking at.

You may include prompts for the user, indicating the kind or range of data to be entered into a specific field. These can be designated Display Only, so that they will be omitted when the form is printed out.

Color is another essential element of form design. The most obvious use is color coding. This can be applied throughout an application: data entry screens in green, browsing screens in blue; or, index key fields in red, ordinary fields in black. Within a particular screen, related fields may be placed within boxes with different color backgrounds and borders.

There are a number of pitfalls in form design. Too many lines, boxes, areas and colors just confuse the user. Go for as simple a scheme as possible. Garish color contrasts attract the eye too strongly. Objects should always be aligned: a group of twenty fields all slightly askew is very hard to read.

Editing Aids

The Form Designer provides a set of editing aids to facilitate design. These aids are available from the Edit menu.

The most frequent editing action is selecting an object prior to changing its characteristics. This is done by double-clicking. See Chapter 1 Getting To Know The Form Designer for more information on selection techniques.

Ruler and Grid

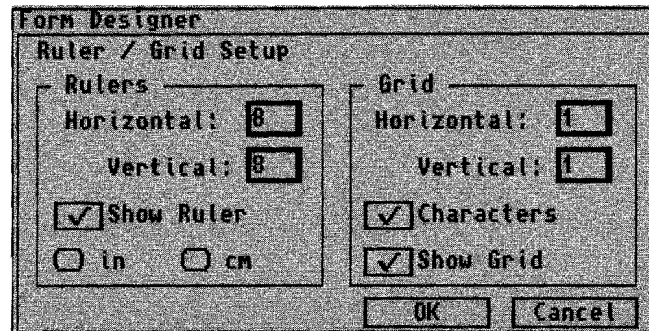
The Form Designer rulers and grid displays are selected with the Ruler/Grid command on the Edit menu. Rulers allow you to ensure that your form layout is correct for the page size on which the form will be printed. One inch on the ruler equals one inch on the paper.

- The ruler as it appears on screen uses logical rather than physical measures. One inch on the screen ruler is not one inch on a real ruler. This is necessary to cope with different screen resolutions, and does not affect the outcome when a form is printed.

The grid is an invaluable tool for ensuring that objects on the form are aligned, both with each other and with precise points on the rulers.

- The grid is hidden under any area, so you may want to place areas on the form last, or use the Crosshairs editing aid to align with the grid outside an area. When editing, resize areas temporarily to expose the grid.

1. Select the Ruler/Grid command.



2. To turn on the rulers, click the Show Rulers check box.
 - ☐ Rulers are displayed in inches or centimeters depending on which option is selected on the requester.
3. Define ruler units in the left-hand box. The units apply to inches or centimeters, whichever is selected.
4. To turn on the grid, click the Show Grid check box.
5. Define grid units in the right-hand box. The values are multiples of the ruler units.
6. You can override the grid units by turning on the Characters check box.
 - ☐ This forces the grid to the character size of the currently selected font. Use this feature with Snap to Grid to position adjoining fields.
7. When you have defined all the settings you require, click OK.

The Form Designer returns to the work area showing your selections.

- To turn off Rulers and Grid, deselect the appropriate check box on the Rulers and Grid requester.

Snap to Grid

The Snap to Grid command forces objects into alignment with the grid. This makes the process of form creation much faster, as you can place objects at approximate positions and the Form Designer will align them for you.

Snap to Grid also affects moved and sized objects.

Already selected objects are snapped to the grid when you choose the command.

Crosshairs

The Crosshairs editing aid allows you to position objects precisely.

When the Crosshairs command is selected, vertical and horizontal lines are shown intersecting at the mouse pointer.

The intersection is shown as X and Y co-ordinates at the right of the toolbox.

When you are using the Report Generator menu commands, the co-ordinates measure character rows and columns rather than physical units.

Auto Field Names

The simplest way to create text prompts for the fields on a form is to use their field names.

When the Auto Field Names option is selected, the name of any field that you place on the form is positioned next to it.

If possible, the Form Designer places the name at the left of the field. If this would entail a collision with the left margin, the name is placed above. If names end up overlapping, they may be easily selected and moved, as they are simple text objects.

Reduced View

If your form is larger than the window, you may wish to see it scaled down to check that the parts are correctly arranged.

Select the Reduced View command to display a scaled down view of the form.

It is not possible to do any editing in the reduced view, and most menu commands are disabled.

Placing Objects on the Form

The order in which types of object are placed on a form is largely a matter of individual preference. There is no reason why you should place fields before text, or the other way round. In this chapter, we present the order of graphic objects, fields, and text.

The general technique for positioning objects is to place them on the form, then move them into the exact position you want. This is quicker for the beginner than trying to get the position right first time, though with experience you will find that your accuracy improves. Of course, the Snap to Grid command helps accurate positioning a lot, but there are always objects that you do not want snapped to the grid.

Defining the Object Hierarchy

The Hierarchy command on the Page menu allows you to alter the existing hierarchy of objects. When you place a new object on top of another object, the Form Designer decides whether to display it in front or behind the other object according to the following hierarchy (from lowest to highest):

- Areas
- Boxes
- Lines
- Images
- Text
- Fields, Commands, Calculations, Functions

The further down the list an object is, the higher the priority it is given. Thus fields appear in front of areas, and text is displayed on top of images. Note that commands include pushbuttons, radio buttons, and checkboxes.

The ability to redefine the object hierarchy can be particularly useful when you want to design a form where the user can select specific points on an image; for example, if the form showed a map of England, clicking on a city might display further information about that city. The simplest way to achieve this effect would be to create a pushbutton for each of the cities in the image. But because pushbuttons have a higher priority than images in the hierarchy they would normally be shown on top of the cities. The answer is to redefine the object hierarchy so that fields have a lower priority than images; i.e., they appear further up the list. This would ensure that the image is displayed on top of the pushbutton. (Another solution would be to retain the default hierarchy but make the pushbuttons invisible – see Chapter 9.)

To alter the position of an object in the hierarchy:

1. Select Page Hierarchy.
 - ☐ The Form Designer displays a dialog box listing the default object hierarchy (as shown above).

2. Select the object type whose position in the hierarchy you wish to change, by clicking it once.
3. Click once on the object currently at the position in the hierarchy to which you want to move. The Form Designer moves the first object to the new position.
4. Repeat this procedure for any other objects in the list which you wish to change.
5. Click OK.

Once you have redefined the hierarchy, the Form Designer saves it with the current form.

Placing Graphic Objects

The graphic objects are described in Chapter 2. The procedure is the same for areas, lines, and boxes:

1. Choose the required object type in the toolbox.
2. Choose color and pattern as required.
3. Click where you want the object to appear.
4. Drag the object to the required size.
 - Areas and Boxes may be square or round cornered.
 - Areas may have borders.

Once an object is on the form, it may be easily sized or moved. See Chapter 2 The Toolbox for more details of these functions.

Placing Text

Text is mainly used to identify objects and applications, and to prompt the user. As a general rule, define the characteristics of text before you place it on the form.

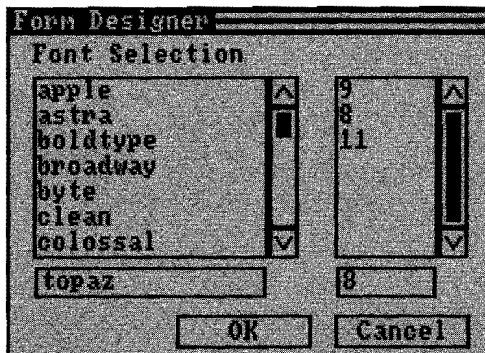
1. Choose a font.
2. Choose a text style.
3. Choose pen color.
 - ☐ Choose a paper color and select the pen and paper tool if you want the text to be drawn with a background.
4. Choose the Text object type in the toolbox.
5. Click where you want the text to appear.
6. Type the text and press ENTER to finish.

Text objects are restricted to a single line. You may easily change the characteristics of text and edit it. See below in this chapter.

Selecting a Font

The Define Font command allows you to choose a font for any text object on the form.

1. Select the Define Font command.



2. Select a font from the list.
 - ☐ The default list of fonts shows all the fonts that are present in the fonts directory.
3. Select a size.
 - ☐ You may type any size into the size box. The Form Designer generates a font as close to the requested size as possible, even if it is illegible.
4. Click OK.

All text, fields, and calculation names will be drawn in the selected font.

Placing Fields

Placing a field on a form allows you to enter data into database records and retrieve it with the same form. You can select fields from any existing database file, and position them anywhere on a form.

Field data has the same characteristics as text data. You may define font, text style, and pen and paper colors for each field. See the previous section.

Fields may be left, right, or center justified. You may designate a field as Read Only to protect it from being edited. Fields, like other objects, may be designated as Display Only, to stop them from being printed. See Chapter 2 The Toolbox for descriptions of these features.

1. Open the database file with the SB Project Open command.
 - ☐ Repeat this step for each file.

2. Select the Field object type (FLD) from the toolbox.
 - ☐ Select the Auto Field Names command on the Edit menu if you want the field name supplied automatically as a text prompt for the field.
 - ☐ Click the Borders tool in the toolbox if you want each field to be supplied with a border in the current border color. The border is always drawn in the finest line pattern.
3. Click where you want the field to appear.
4. Select a field from the list shown.
 - ☐ You may switch to another file by clicking the arrow box next to the file name on the requester.
 - ☐ If you place fields from more than one file on a form, you should link the files before saving the form. See Chapter 6 Linking Files.
5. Click OK.

A dotted box containing the field name is placed on the form at the point where you clicked.

The size of the box is determined by the font and text style. You should resize long text fields to a rectangular shape.

External File Fields

If you wish the form to show external images or text in an external file field, you must use the size tool to make the field box large enough to hold the image.

- **Clipping.** If an External File Field is left justified, the top left-hand corner of the image will be shown on the form. Center and right justification show the center and bottom right-hand corner of images respectively.
- **Scaling.** If you select the Image Scaling tool in the toolbox, Superbase will scale PCX images to fit the size of the field box.

Fields from dBase Files

In Superbase, you may open dBase files for browsing, query, and output. dBase files may not be modified from within Superbase, so there is no risk of affecting existing data. Commands which would write to the dBase file are disabled.

You may also import dBase files into Superbase so that you can use all the menu commands.

Before you use a dBase file in the Form Designer, you must create a special definition file in Superbase itself. Open at least one index for the dBase file before using the Superbase Project Save command.

In the Form Designer, select the SB Project Open command and click the dBase check box. Any dBase DBD files in the current directory will be listed, and can be opened in the usual way.

Note that you may use the Superbase Project Modify command to alter the display format of individual fields.

Saving Forms

When you have completed the design of your form, select the Project Save As command to save it on disk. The Save command saves an existing form under its own name, whereas Save As allows you to give it a new name.

- You can also save individual pages from a form to build up a library of form designs. See Managing Pages later in this chapter.
- You must save a form after editing it for Superbase to detect that it has changed and re-open it automatically.

See Chapter 10 Generating Reports for a description of what happens when you save a report form.

Editing Forms

Whenever you make changes to a form, you are editing it. The Form Designer provides many tools and commands so that you can easily adapt a form by changing its graphical layout or logical contents.

The toolbox at the bottom of the window is the designer's main aid. You select an object on the form, then change its color or pattern or another characteristic by choosing the appropriate tool from the toolbox.

Other ways of editing a form include the use of the cut and paste commands, and the tools for moving and resizing objects. See Chapter 2 The Toolbox for details on Move and Size and other tools.

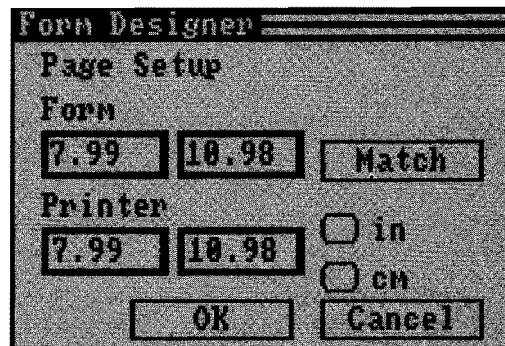
Adjusting Form Page to Match Printer Page

The editable area is taken from the dimensions of the page set with the Printer Setup command. Sometimes there will be a difference between the dimensions of an existing form and those currently set in the Form Designer.

If the printer page size is smaller than the screen form size, a broken line will appear on screen to show the dimensions you must work within.

If you have opened an existing form the printer page dimensions may be different from the screen form dimensions.

Select the Printer Setup command from the Project menu, select the printer you intend to use from the Change Printer requester and click on OK. The Page Setup requester is displayed.



The requester shows the current settings for the selected printer. If you want to change any of these, set the new values before clicking on OK.

If the Form Designer detects any objects partly or wholly outside the new boundaries, it selects all such objects and advises you that the adjustment cannot be completed. You must then move the selected objects inside the boundaries, or reset the printer page dimensions.

Selecting Objects

In order to change any characteristic of an existing object, you must first select it. On the screen, a selected object is shown with a border of dashes.

Select an object by positioning the mouse pointer over it, then double-clicking the mouse button.

You may select more than one object at a time. To select other objects singly, hold down the SHIFT key while clicking them.

If the objects are in a group, hold down the CTRL key and hold down the mouse button. This produces an expandable selection box, which you drag until it fully or partly encloses the objects you wish to select. When you release the mouse button, all the objects are shown as selected.

To deselect an object, click it, select another object, or click on an unoccupied part of the form.

Select also Chapter 1 Getting To Know The Form Designer.

Undo

The Undo command reverses the previous editing action. Actions that may be undone include Move and Size. Undo also removes the last object placed on a form if your last action was to create it.

Undo does not reverse the commands on the Define menu, or the Page Clear or Erase commands.

Cut, Copy, Paste, and Clear

The Cut and Paste commands work in the normal way. First, select the object or objects to be edited.

Next select the command you require from the Edit menu.

Cut

Removes the selected object and stores a temporary copy.

Copy

Stores a temporary copy of the selected object.

Clear

Removes the selected object from the form.

Paste

This command works differently. First you select the Paste menu command, then you click where the object is to appear. You can then move the paste box to the required position and release the mouse button. The Form Designer inserts a copy of the stored objects where you clicked.

- Once an object has been stored with Cut or Copy, it may be pasted many times.
- See also the Page Clear and Page Erase commands, described below.

Editing Text

Text objects may be edited in two senses: by changing the characteristics of the object, or by changing the content of the text.

To alter the way a text object is drawn, select the text to be edited as you would any object. You may select multiple text objects if you wish.

Now you may vary the color and other characteristics of the text simply by clicking the appropriate tools in the toolbox.

The font of selected text may be altered by choosing a new font with the Define Font command.

To change the content of a text object, select the Text object type and then click the text object. An insertion point appears at the beginning of the text. You may now delete and retype the text. Press ENTER to finish.

Changing File and Field References

The field names used in a form are taken from a Superbase file definition. When you open an existing form for editing, the Form Designer checks to see whether the Superbase file is still present in the directory. You may have reorganized the file from within Superbase, giving it another name, or you may have created an empty copy and removed the old file. Your form would still be valid if you could change the filename for all the fields from the original file to the filename of the new file.

If the file is not there, the Form Designer informs you that it cannot find the file, and presents you with a Replacement File requester.

Select the new file. All references to the old filename – the one that was not found – will be updated.

Similarly, you may have changed a field name in Superbase. When you open a form that still has a reference to the old field name, the Form Designer informs you that it cannot find the field, and shows a list of the fieldnames in the new file definition. Select a field to replace the one that could not be found.

If the Form Designer finds a reference that needs changing in a calculation, a validation, or a command, it presents the appropriate requester for you to edit the command line or formula. If you click Cancel the object will remain but the formula or command line itself will be cleared out.

- Remember to save the form.

Managing Pages

Forms may consist of one or more pages. If you have to design many pages, you can build up a library of pages on disk from which to assemble more complex forms. This can greatly reduce the time required to develop input and browsing screens, especially if you adopt formal standards so that the forms used in different parts of an application look similar.

The Page menu includes commands for creating a new page, opening and saving pages, clearing and erasing pages, and for switching between pages.

New Page

When you want to create a second or subsequent page for your form, use this command.

It starts a new page after the current page. You may define the page dimensions of each page separately.

Opening and Saving Pages

Individual pages may be opened and saved with the Page Open and Page Save As commands.

Page Open lists all PG files in the current directory. Opening a page file causes it to be inserted as the current page of the form.

Page Save As allows you to save the current page of the form as a PG file.

Page Clear and Page Erase

If you want to remove all objects from a page, use the Page Clear command.

In a multi-page form, you may need to remove a page completely. The Page Erase command deletes the current page.

You cannot restore a page that has been erased or cleared in this way with the Undo command, so you may wish to save the page separately first.

Page Next / Previous / Go to ...

During the design process, you will need to move from page to page. The Page Next and Previous commands switch to the neighboring pages of the form. The Page Go To command presents a requester allowing you to specify the required page.

See also Chapter 12 Housekeeping.

4 TRANSACTION LINES

Many applications include one-to-many relationships between records in different files. In a form, you can define a group of fields as the 'many' part of such a relationship. The term 'transaction lines' refers to such a group.

Typically, each transaction is a row of fields from a single record belonging to the file at the 'many' end of a one-to-many relationship.

Once a block of transaction lines is defined, you can enter several data records into the same file through one form. Provided all the files in the form are linked, Superbase will maintain the relationships between the transaction line records and records in other files. This allows relational browsing, in which a form shows details from many associated records at one time.

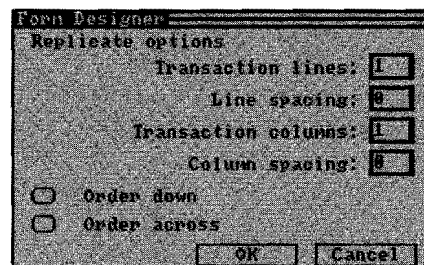
To define transaction lines, follow this procedure:

1. Place all the fields that are to be part of each line on the form.
2. Select them.
3. Choose the Define Transaction Lines command. On the requester, you specify parameters for the command:
 - ☐ The number of transaction rows you want.
 - ☐ The number of blank lines that are to appear between transaction lines and rows.
 - ☐ The required arrangement of columns and rows.
4. Click OK.

The Form Designer creates the block of transaction lines.

- You may only create one block of transaction lines on a form.
- Provided the files are linked, only one of the common reference fields need appear on the form, and it need not appear in the transaction line block.

The Transaction Lines requester



Form Designer

Replicate options

Transaction lines: 1

Line spacing: 8

Transaction columns: 1

Column spacing: 8

☐ Order down

☐ Order across

OK Cancel

Transaction lines specifies how many times the selected group of fields and calculations will be replicated down the page: the number of transactions that will appear on the form.

Line spacing determines the number of empty lines between transactions down the page. Enter a value of 0, if you wish the transactions to be placed without any spaces between them.

- Row and column spacing increments are taken from the size of the largest font used for an object in the group being replicated.

Transaction columns allows you to specify whether the transactions are to be arranged in a single column or across the screen in two or more columns. The total number of transactions on the form will be the number in this box times the number in the transaction lines box. If you want the form to have just one column of transactions, enter a value of 1.

Column spacing only needs to be specified if you are using more than one transaction column. The number in this box determines how many spaces will be placed between each transaction across the page when there are more than one to a line.

Order Down/Across buttons determine the entry order for transactions. If the form has only one column of transactions, they will automatically be numbered from top to bottom. With multiple columns, you have the option of setting the entry order from top to bottom (Down) or left to right (Across).

1	5
2	6
3	7
4	8

Clicking the Across button would ensure the entry order was as follows:

1	2
3	4
5	6
7	8

Note that this command only determines the entry order for the transaction line as a whole. It doesn't affect the order of the fields within a transaction.

Editing Transaction Lines

Despite their greater complexity, transactions can be edited in the Form Designer in the same way as other objects on the form. Editing a transaction line includes deleting fields, adding new fields, and redefining the arrangement of a transaction block by selecting Define Transaction Lines again.

To change the number of lines, first reselect the block.

- This is not strictly necessary, but note that all selected fields will be included as part of the transaction lines.

Now reselect the command and specify the required number of lines.

Other editing activities treat the transaction lines block as a unit. So, if you change the color of a the third field along, all corresponding fields will also change.

Before editing a transaction, you need to select the item or items you wish to change. This has the effect of selecting all the items of the same type in the transaction block; i.e., selecting a field in the first line automatically selects the same field in the other lines.

To add a field, first place it on the form, select it in the usual way, and then select Define Transaction Lines. It is not necessary to select the fields that are already defined as transaction elements; after you have redefined the transaction block using Define Transaction Lines, the new field will be incorporated in the transactions.

A Transaction Form Example

Consider an example of a database application for cataloguing a book collection. It uses two files, **Authors** and **Books**. The **Authors** file stores the name and other details of all the authors represented in the collection, using one record for each author. The records in the **Books** file store the details of each book, such as its title, subject matter, and publisher. In addition, the records in both files contain an alphanumeric code (the **Author_code** field) which links the books to their authors.

This way of organizing the data is far more flexible than a system which uses a single file to catalogue the books. It makes it easy to design a form which displays the authors and a list of their books on screen at the same time. The data would be structured as follows:

Author's name, Author code
Title, Subject, Publisher
Title, Subject, Publisher
Title, Subject, Publisher

Every time you selected another record in the **Authors** file, Superbase would read in the next author in alphabetical order followed by a variable number of lines, one for each of the author's books. The fields from a record in the **Books** file are treated as a single transaction; a set of transactions represents the books written by a particular author.

It is possible to include fields from other files in the transaction line; in this example, each transaction line could show the publisher's address by looking it up in an address file. You would not want to amend the publisher's details from this form, so you would make all the fields from that file Read Only.

Calculations in Transaction Lines

A transaction line may also contain form calculations. These must be tied to the fields in the line; in other words, the calculation formulas must include a reference to one or more transaction fields.

Calculations are replicated at the same time as the fields. You simply select them as part of the multiple group.

The Form Designer turns repeated calculations into an array. Remember that a form calculation can be treated as a program variable. When you replicate a calculation it becomes an array variable; for example, replicating the calculation `tot% five times` creates an array with five elements, `tot%(0)` to `tot%(4)`.

Note

If the calculation sends its result to a field, the field must be one that occurs in the transaction line. It should not post data to fields in other files.

Linking Files

For a transaction form to work, you must establish a link between a field in the master file and a field in the transaction line using the Link command on the Set menu. Thereafter, you can set other links, but these must always be between fields in a 'one-to-one' relation. In other words, you cannot set up forms where a transaction has further transactions of its own.

See Chapter 6 Linking Files.

Setting the Data Entry Order for Transactions

Chapter 5 Defining Data Entry Order explains the procedures for determining the sequence used for moving from field to field during data entry. This section provides additional material relevant to transaction lines.

The Define Data Entry Order command allows you to define the data entry order for the fields and calculations in a transaction in the same way as for non-transactional fields. However, there are a number of restrictions that apply when you are setting the entry order for the elements of a transaction.

- The data entry order within a transaction is the same for all the transactions on the form. If you specify for one transaction that data is entered from left to right, this will also be the case for the other transactions. This means that when you are setting the entry order, you should concentrate only on the first transaction. Any changes you make will then be automatically repeated in the rest of the transactions.
- If you alter the order of a transaction element with respect to other fields on the form, all the transaction order numbers will be altered at the same time (although they remain the same relative to each other).

To illustrate these points, let's consider an example involving a form with four transactions each containing three fields. In addition to the transactions, there is one non-transaction field on the form above the transaction block. The fields are arranged like this:

	Field_a	
Field_b	Field_c	Field_d
Field_b	Field_c	Field_d
Field_b	Field_c	Field_d
Field_b	Field_c	Field_d

Field_a stands on its own; the other fields are elements in the four transaction lines. Initially, the entry order is as follows:

	1	
2	3	4
5	6	7
8	9	10
11	12	13

Suppose you wanted to change this order so that the user was required to enter data in the second field (**Field_c**) in each transaction line before the first field. To do this you would click once in **Field_c** on the first transaction line, and then again on **Field_b**. The change takes place after the two fields have been selected.

The entry order would now be:

	1	
3	2	4
6	5	7
9	8	10
12	11	13

You might also want to ensure that data was entered in the transactions before **Field_a**. In this case you would need to select the Data Entry Order command again, and then click twice on **Field_c** in the first transaction.

The entry order would then be:

	13	
2	1	3
5	4	6
8	7	9
11	10	12

Copying Fields and Calculations

Transactions may only be created by repeating fields (and calculations) with the Define Transaction Lines command. If you repeat a field from a transaction line

using some other technique – by copying it or by adding it to the page a second time – it will only show the data for the current record and the current transaction line.

Usually, this field would be marked as Read Only and would be used to display a full description of data which is truncated in the transaction line.

Using Report Functions

The report functions can be applied to the fields and calculations that occur in transactions. They are used in formulas which are attached to calculations outside the transaction block. An example, would be the formula:

SUM Annual_interest

This returns the total value of the field **Annual_interest** in all the transaction lines that contain data. Note that the report functions do not operate on empty transaction lines.

Other examples are:

COUNT Stockcode
MEAN
MAX
MIN

Apart from COUNT, the report functions can only be used with numeric fields and calculation variables.

Report functions can also be used in DML programs. As well as working with transaction items, they can be applied to the arrays which are defined within a program. See the entries for these functions in the DML User Guide.

Programming Transactions

One of the advantages of using a form to process transactions is that it does not require any programming. However, transaction forms on their own will only take you so far. For more complex applications, you will need to combine the form's built-in processing facility with a program. Superbase provides the following commands for handling transactions:

SELECT FORM
UPDATE FORM
ENTER

When they are used with the ROW keyword, these commands give the programmer control over input and output to the individual lines in a set of transactions. For more details, see the DML User Guide.

Creating a Table Form

If you define a form to show just a set of transaction lines from a single file – no one-to-many relationship – you can edit any of the visible transactions simply by clicking in it.

A form like this will display multiple records from one file in a similar way to Superbase's Table View command. But it has the advantage of allowing you to click on any of the lines on screen. The line then becomes the current record. You can use this feature for two purposes: as a way of selecting a record from a list, and as a way of entering data into multiple records at the same time.

5 DEFINING DATA ENTRY ORDER

You can define the sequence to be followed when you enter data into a form. This allows you to follow logical sequences without being tied to an inflexible structure. You may even jump between different pages of a form if you wish.

The Define Data Entry Order command allows you to define the order in which data is entered into fields, calculations, and commands.

You should wait until all fields, calculations, and commands have been placed on a form before you use the command, as adding or removing fields resets the data entry order to the left-to-right, top-to-bottom default.

By default the data entry order is the order in which you add fields to a form. For example, if you add a **Firstname** field before a **Lastname** field, users will be required to type in their first names before typing their last names. Using the Data Entry Order command, you can instruct the Form Designer only to accept data in the order you specify, irrespective of how the fields were added to a page and where they are positioned on the page. The data entry order applies to all the fields in a form, so you can also define an order which extends over more than one page.

When you select Data Entry Order, the Form Designer redisplay the form with the names of all fields, calculations, and commands replaced by numbers.

Date	D's Ref	Client Name	Quantity	US\$ Value	Commission
65	0 00	01	07	08	02
73	0 07	06	74	79	03
80	0 04	05	81	82	84
87	0 01	82	88	89	83
94	0 09	89	95	96	000
101	0 05	104	106	103	102
106	0 12	113	109	110	114
115	0 11	120	116	117	121
122	0 12	127	123	124	126
129	0 13	134	130	131	135
Transaction Page Totals:				146	147

You are presented with a requester which gives you a choice of resetting the current order or leaving it as it is. If you select Reset, the fields are numbered from left to right on the same line, then line by line, and page by page:

Field1	Field2	Field3
Field4	Field5	Field6
Field7	Field8	Field9

You can now define a new data entry order by clicking the fields and calculations in the required order.

Clicking once in a field makes it the current field and does not reset its order number. Then you define the data entry order for the remaining fields by clicking in each field once.

1. Decide which field you want to start from.
 2. Select it as the current field by clicking in it once.
 3. Decide which field comes next in order.
 4. Click once in the field to make it the next field in the data entry order.
 - The field you click in becomes the current field.
 5. Continue clicking in fields until you reach the last field, or until the order is as you want it.
- You must save the form to preserve the defined order.
 - During data entry, control passes through a read only object immediately, creating an insertion point in the next non-read only object in the sequence. This is very useful for forcing calculations and commands to be executed at the correct time.
 - Click twice (but do not double-click) to take a field to the top of the list – to make it the first field.

Example

We can illustrate by taking the data entry order shown above as an example. Suppose the first three fields you clicked on were **Field2**, **Field5**, and **Field9**. The new field data entry order would then become:

Field1	Field2	Field5
Field6	Field3	Field7
Field8	Field9	Field4

After the first click, clicking on a field automatically changes its order number and, at the same time, makes it the current field.

Note

If a field is only slightly higher on the page than another, the Form Designer will regard it as being on a different line. When you reset the data entry order the field on the higher line will be placed before the field on the line below, despite the fact that it may appear to be on the same line, and may be positioned further to the right.

Changing Data Entry Order

When you have many fields on a form and you wish to alter the order of just some of them, follow this summarized procedure:

1. Select Define Data Entry Order.
2. Do not reset the order.
3. Choose the field before the first field that forms part of a new sequence and click it.
4. Click all the fields in the new sequence in turn.
5. When no more fields need to be changed, save the form.

If you need to change more than one sub-sequence within the main sequence, you must select the Define Data Entry Order command for each one.

—

—

—

6 LINKING FILES

Overview

When you design a form with fields from more than one file on it, you must use the Define SB File Link command to set up the information that Superbase needs to process the form correctly. The links between files on a form are known as the Link Structure of the form.

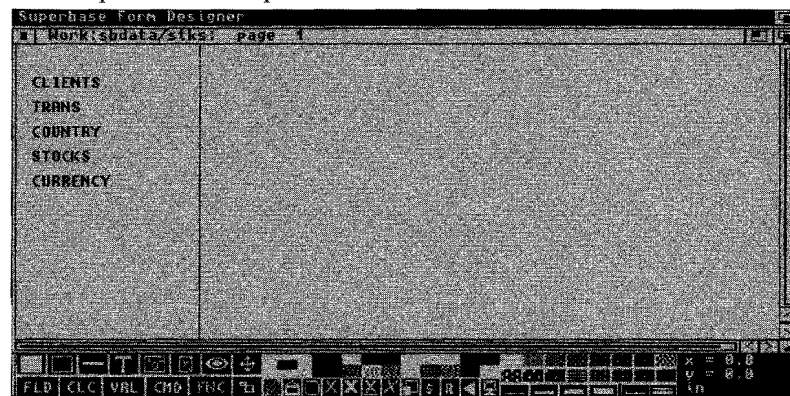
The Link Structure does not replace the LOOKUP validations that exist in the file definitions of a multi-file application. It is not used directly to regulate the entry of data into multiple files. The Link Structure mainly defines the way in which a given form retrieves sets of related records during browsing. Remember that it is possible to have many different forms for any one file or set of related files.

In Superbase, any file that is part of the Link Structure of a form may be set as the current file. You may then step through the records of this file in the usual way, and Superbase will retrieve the records from associated files that were originally entered on the same form.

Because the Link Structure ensures that all related records are in memory at the same time, it automatically allows you to edit existing data with a form and be sure that all the relationships between fields in different files are maintained as they were when the form data was created.

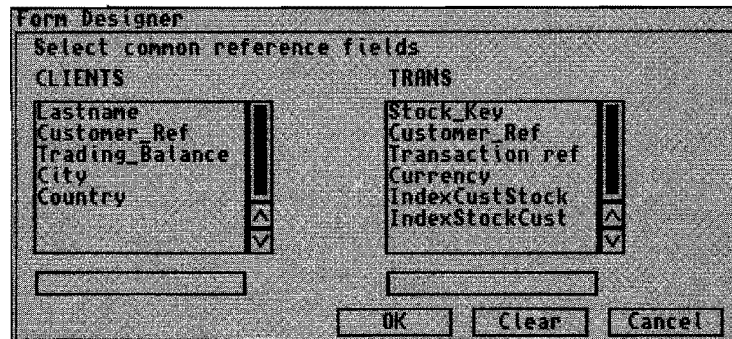
Linking Files

Wait until you have added all the fields required for the form, then select the Define SB File Link command. The display that replaces the normal screen is used to build up a schematic representation of the links between the files.

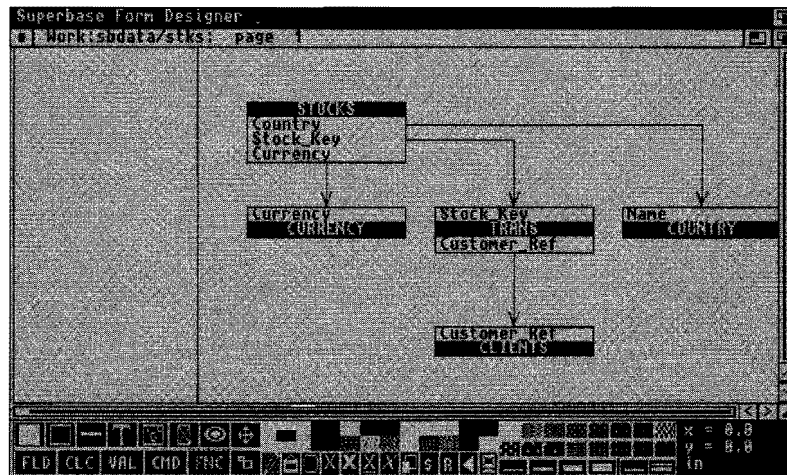


The files available for linking are listed in the left-hand panel.

- A file may only be added to the Link Structure if at least one field from it is placed on the form or referred to in a calculation.
1. Click the name of the file that will be the Master File.
 2. Click the first file to be linked to it.
 3. Click on the Master File box in the Link Structure.



4. On the requester, highlight the Common Reference Fields that will form the link and click OK.
5. Click the next file to be linked to the Link Structure.
6. Click on the file box in the Link Structure to which this file is to be linked.
7. Specify the Common Reference Fields.
8. Repeat steps 5 through 7 until there are no files in the left-hand panel.



Master File

The Master File is the file that Superbase uses to control the default browsing order of a multi-file form. In many situations where the form is based on a one-to-many relationship between two files, the master file is the 'One.'

Common Reference Fields

Every pair of files to be linked must have a pair of indexed fields that contain matching data, such as an account or product code. When you set up the file definitions in Superbase for files that will be used for relational browsing, ensure that fields to be used as Common Reference Fields have the same attributes, such as uppercase conversion.

See the chapter on Multi-File Applications in Volume 1 for a fuller discussion of this topic.

Adding a File to the Link Structure

The Common Reference Fields for a pair of files may require you to add a file to the Link Structure at a point within it.

When you go to add a field to the Link Structure, first click the file you want to link and then the file to which you want to link it. After you have specified the common reference fields, the Form Designer adjusts the Link Structure accordingly.

Editing a Link Structure

You may edit an existing Link Structure by selecting the Define SB File Link command.

The Form Designer shows the Link Structure. To change a link at any point within the structure, you must clear and redefine it.

1. Click the file box of the file to be unlinked.
 - ☐ This shows the requester with the Common Reference Fields highlighted.
2. Click Clear.
3. Click OK.

All dependent links will be deleted from the Link Structure.

- Any file that has been unlinked is shown in the left-hand panel.
- To clear all links, follow this procedure for the Master File.

You may now set new links as described above.

—

—

—

7 FORM CALCULATION FORMULAS

Calculations allow you to design forms which have their own built-in data processing facility. By means of calculations, forms can be used in Superbase not only to display record data but also to carry out many of the tasks that would normally be left to a program. They can calculate results on existing data and they can operate on new data as you enter it.

As the next section describes, there are many different types of calculation. Some of them can be regarded as special types of program variables; you will use these to store the results from other calculations. Other types of calculation have formulas attached to them and are closer to calculated fields in a Superbase file. The calculations in a form, however, are not directly attached to fields, but stand as objects in their own right; and, unlike fields, they are not stored in Superbase files but exist only in a form.

Commands are created in a similar way to calculations, but contain sequences of DML statements. See Chapter 9 Commands.

Creating a Calculation

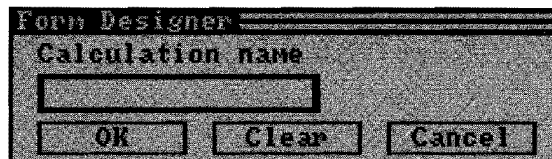
The process of creating a calculation is very similar to defining a calculation formula for a Superbase file definition. The main difference is that the calculation has to be given a name so that it can be referred to by other calculations or by programs.

1. Choose the Calculation (CLC) object type from the toolbox. 2.

CLC

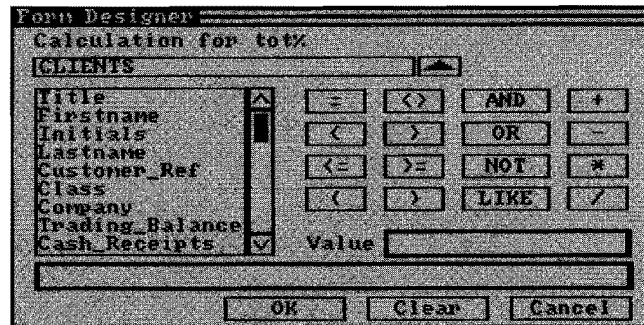
Click where you want the calculation to appear.

- ☐ You may want to use a grid or the Snap to Grid or Crosshairs editing aids to help you position the calculation accurately.
3. Enter a name ending in \$ for calculations that will hold a text or string result, and % for all other types.



- ☐ Do not use any Superbase reserved words. (See Volume 1, Appendix C.)

4. Click OK.



5. Enter the formula for the calculation in the Calculation requester, unless it is a Variable Calculation (see below).
 - The requester will show a list of the fields (in the current file) which have been added to the form. If you want to refer to other calculations in the formula, click on the arrow button next to the file name. This cycles through all the available files. After the field lists the Form Designer shows a list of the names of all calculations already defined as if they belonged to a file called **Formcalcs**.
6. Click OK.

The Form Designer positions the calculation on the form. You may now size or move it.

- You may refer to fields that do not actually appear on the form.
- You may refer to other calculations provided they have already been placed on the form.
- A calculation name should only appear in the calculation if it is intended to be self-referencing. You can only incorporate the name of a calculation in its formula after it has been created and placed on the form.
- The Form Designer resets the default length of a calculation every time it is modified.

Formula Calculations

Formula calculations are the same as the calculations usually attached to fields in a file definition. See the chapter on Derived Values in Volume 1 for a full discussion of this topic. They may be of use on a form, but the results will not be stored in a database field unless you include extra instructions (see Evaluation Control Prefixes, below, and Chapter 9 Commands).

Formulas combine field names, functions, operators and literal values to generate results, which are displayed in the calculation area.

Typically, this type of calculation is used to derive a total from the contents of other calculations or fields and then to display it on the form. Another application would be to display a date by using the keyword TODAY in the formula.

Examples are:

Calculation name	Calculation formula
subtot1%	Quantity * Price
name\$	Left\$(Firstname,1) + ". " + Lastname
When\$	TODAY

Formula Evaluation

During data entry, a formula is evaluated and the result is displayed after each of the following events;

- Saving a record.
- Moving the insertion point through the calculation by pressing ENTER in the field that precedes it in data entry order, assuming the calculation is read only.
- Clicking the calculation, provided it is read only.
- When you press ENTER after typing in the calculation, provided it is NOT read only.
- Retrieving another record.

Variable Calculations

These are calculations which do not have a formula attached to them. They are intended to be treated as variables and provide an interface to DML programs which process forms.

A variable calculation may have data typed into it by the user; the data may then be accessed by a program.

Suppose, for example, you define a blank calculation with the name x%. In the form's data entry order x% is number 5. If you want to input data to a program from the form which contains x%, you must include the program line:

ENTER 5

When this line is executed, the insertion point appears on screen in the box belonging to x% and the value entered by the user is assigned to x%.

If you now executed the MEMORY command, x% and its contents will be listed along with all the other program variables. And any other DML commands which operate on variables, such as CLEAR, will also work with a calculation variable.

Note

ENTER is the only input command that can be used with a form. Forms are object oriented rather than character oriented so you cannot use ASK or GET in combination with LOCATE. Another point to note is that when you use ENTER with a calculation, you can only identify the calculation by its field order number; a command such as:

ENTER fred%

will not work.

Assignment Calculations

This type of calculation is used to assign a value to a field or another calculation. The calculation requires the keyword LET at the beginning.

An assignment calculation has two parts: an assignment and a formula. For example, the calculation y% may contain:

LET fieldA.FILEA = fieldB.FILEB * 1.15

Here the right-hand part of the calculation will be updated and displayed in y% in the normal way. The result will be assigned to **fieldA**, which may belong to another file and does not have to be placed on the form.

When Superbase reads a new record – for example, when you select a record with the browsing controls – the right-hand part of the calculation is evaluated and displayed, but no assignment is made unless the calculation is evaluated as a result of editing or saving.

Only one assignment per calculation is allowed. Multiple assignments may be achieved with a command.

Evaluation Control Prefixes

Two special prefixes may be used with Assignment Calculations and Formula Calculations to vary the way they work.

AFTER

The AFTER keyword causes a formula to be evaluated more frequently than usual. You may use this feature to force evaluation after any editing event; the formula

AFTER : Quantity * Price

will be evaluated and the result displayed whenever data changes on the form.

By including a field reference the evaluation may be triggered by an editing event for that field:

AFTER Item : Quantity * Price

Now a result will be obtained when Item is edited or its value changes, as well as after the usual evaluation conditions.

POST

The POST keyword restricts evaluation of a calculation to when data is saved. This allows you to ensure that data is only assigned to a field once, after all the other calculations in the form have been evaluated.

For example, you might wish to transfer data from an invoice form to a balance field in a completely different file. You would only want to do this once, not every time data was edited, so the syntax for a calculation z% should be:

POST LET Balance.CUSTOMER = Balance.CUSTOMER +
Total.ORDER

The contents of **Total.ORDER** will only be added to **Balance.CUSTOMER** when all the other form calculations have been performed and the record is saved. The intermediate results will be shown whenever the formula is evaluated.

Note

If you are using an Assignment Calculation to send or 'post' data to a record in another file, remember to save the record after completing a task.

8 FORM VALIDATIONS

If you are dealing with important information, you need to be confident that stored data is accurate. The only time you can control the quality of your data is when it is first entered. Validation is a way of ensuring that the data entered into a field meets certain criteria.

Validation works in the same way as it does in Superbase when it is used in a file definition. It allows you to assign a validation formula to a field. The formula sets the limits to what you can enter in the field.

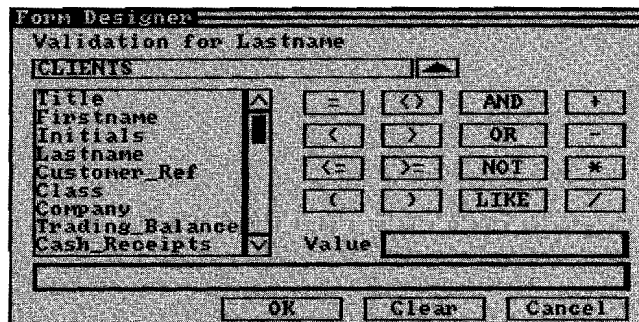
Validations on forms are intended to complement the main validations for your application, which will be permanently associated with your file definitions. This is especially true of cross-file LOOKUP validations, which should not normally be placed on a form. Form validations allow you to apply extra validations which are relevant to the part of the application with which the form deals. Different users will have different activities. For example, you might wish to limit a sales assistant's scope for editing but allow a senior manager full access to data.

Superbase automatically validates the type of data that goes into date, time, logical, and number fields. You can also define your own formulas to validate the contents of any field. Field data is checked against its validation formulas when you enter data into fields or save a record.

Creating a Validation Formula

The field to be validated must already be placed on the form.

1. Choose the Validation object type (VAL) from the toolbox.
2. Click the field, and the Form Designer displays a requester with which you can build up a validation formula.



- ☐ You may not add a validation to a form calculation. However, you may use a command to test the result of a calculation.
 - ☐ Validation formulas do not appear as objects on the form. However you may change them as explained below, and they are listed in the output of the Form Status command.
3. Construct the validation formula. The formula always refers to the field being validated. For example, if the field was **Field1.FILEA** and it was restricted to the values 'A' or 'B' the formula would be:

Field1.FILEA = "A" OR Field1.FILEA = "B"

Click the field name in the fields list box, then click an operator, then type in a value and press ENTER. You can extend the formula in many ways.

- ☐ Combine fields in complex arithmetic expressions.
- ☐ Use functions to tap the power of Superbase's DML language.
- ☐ Use pattern matching to allow a range of choices.
- ☐ Add your own help message:

Field1.FILEA = "A" ELSE "This field must equal 'A'"

Every validation formula must yield a result which is true or false. A formula such as **Field1.FILE + Field2.FILE** would be incorrect as a validation.

4. When you have completed the formula, click OK.

Changing a Validation Formula

1. Choose the Validation object type (VAL) from the toolbox.
2. Click the field to which the validation belongs, and the Validation requester appears. Make any necessary changes.
3. Click OK.

9 COMMANDS AND CONTROLS

The Form Designer includes facilities for adding commands, pushbuttons, check boxes, and radio buttons to forms, enabling you to create sophisticated and complete applications without having to produce a full-scale Superbase program suite.

By placing pushbuttons on forms, you gain access to Superbase's DML programming language, either in the form of self-contained command strings or by executing routines in a memory-resident DML program. Pushbuttons may be used for almost any purpose: for example, to trigger complex processing, to save changes after editing form data, or simply to switch to another form.

Command strings do not have to be attached to pushbuttons. You may need some processing to be performed automatically during the course of data entry, and to achieve this you can hide a command object within a form so that its presence is not apparent to the user.

A command is one or more DML statements entered as a single line. The command must follow DML syntax as defined in the DML Reference Guide.

In addition to pushbuttons and commands, Superbase provides two kinds of form control: radio buttons and check boxes. These controls are attached to fields or variable calculations.

Pushbuttons

A pushbutton is a graphic object that executes a command. Visually, a pushbutton is a rectangle containing a text string that indicates its purpose. Logically, a pushbutton is a sequence of DML commands. When the user clicks the pushbutton, Superbase executes the command sequence.

To define a pushbutton:

1. Select the Command object type (CMD) from the toolbox.
2. Click where you want the pushbutton to be placed.
3. Enter a variable name such as **comm1\$**.
 - ☐ Do not enter a numeric name such as x% unless you plan to give the button a numeric title such as '100.' An ordinary text title will not be displayed if the pushbutton has a numeric name.
4. Enter the DML statements that comprise the command.
 - ☐ If there is more than one statement in a command line, statements must be separated by colons.

- ☐ The keywords LET, AFTER, and POST should not be used to begin commands; these keywords are used in form calculations.
- 5. Turn on the Pushbutton check box.
- 6. Unless the pushbutton command is intended to be executed automatically – that is, as part of the data entry process – turn on the Skip Entry Execution check box.
- 7. Enter the Button Title. This is the text that you want to appear in the Pushbutton. It is automatically centered.
 - ☐ The Button Title text may be varied while a form is being used, either by DML or by the execution of other form commands.
- 8. Click OK.

Editing Pushbuttons

You may alter the visual characteristics of a pushbutton control by selecting it in the usual way, then choosing the attributes you require, for example:

- Size. The pushbutton may be any size.
- Font and style. The Button Title is drawn in the current font and text style.
- Color. You may vary the color of the border, and draw the Button Title text in any pen and/or paper colors. Borders may be turned off.
- Read Only. Commands are automatically designated read only, but this attribute may be turned off.

If you want to change the contents of the pushbutton command or its title, you must use the Command requester.

1. Select the Command object type from the toolbox.
2. Click the pushbutton to be edited.
3. Change the name of the command if required.
4. Make any changes to the command sequence or button title.

Using Pushbuttons

Pushbuttons may be used to control a form-based application, for example by allowing the user to switch from one form to another, call up lists of queries, and so on. In such a situation, the pushbutton is not part of the data entry process, and should be excluded from the data entry order by turning on Skip Entry Execution. On the other hand, a pushbutton could be used to assign predictable values to several fields in one go, as a way of speeding up the data entry process. In this case you would leave Skip Entry Execution turned off.

To use a pushbutton when a form is open, you may click it directly. This executes the command whether the pushbutton is read only or not.

If a pushbutton is part of the data entry process, and has been made *not* read only, you may tab to it and execute it by pressing SPACEBAR. ENTER is used to move to the next field or control. Read only pushbuttons are executed automatically if they are part of the data entry process.

Commands Without Pushbuttons

The procedure for setting up command strings that are to be executed automatically is almost the same as for pushbuttons. On the Command requester, make sure the Pushbutton check box and the Skip Entry Execution check box are turned off, and do not enter a title.

After creating the command object, adjust its colors so that the text is not visible against the background.

Auto Execution

If a command both comes first in the data entry order sequence and has read only status, it will be executed automatically once only: the first time the form is opened during a database session. This feature may be used to load a DML program consisting of subroutines that are subsequently called by other commands.

When is a Command Executed?

Commands are executed when the insertion point passes through them. If the command is read only, the insertion point will never appear in it, but will pass through on its way from one editable object (field or calculation) to the next. This allows you to place a command that is to be used for validation right after the field to which it relates.

Of course, you must ensure that the data entry order includes the command at the correct point.

See Chapter 5 Defining Data Entry Order.

Interfacing to a DML Program

You may use the keywords RUN, CHAIN, LOAD, GOTO, and GOSUB. This allows you to construct an interface between a form and a set of DML subroutines that are called from the form.

RUN Loads a program from disk and executes it.

CHAIN Loads a program from disk with preservation of variable contents, and executes it.

LOAD Loads a program from disk but does not execute it. This allows you to load a program consisting of subroutines and then access them individually with the following two commands.

GOTO exits to a specified label, and starts executing from that label. It does not return to the form at all.

GOSUB exits to a label, and starts executing the subroutine at the label. Control returns to the form command when a RETURN statement is encountered.

- When control is returned to the form command after a GOSUB, it returns to the next executable statement in the command, assuming that the command has more than one statement. If the GOTO or GOSUB was the last statement in a command, control returns to the next command or field in the data entry order.

END in a command terminates data entry and removes the insertion point from the form.

ENTER in a command that is executed during data entry may be used to redirect data entry to a specific field. ENTER 5, for example, will place an insertion point in field number 5 in the data entry order.

- You can use only literal numbers in this situation, as the use of numeric variables for calculation and command names conflicts with the use of them as arguments to ENTER.

ENTER END in a command terminates data entry. If no DML program is running, the standard 'Save This Form' requester is invoked.

OPEN FORM may be attached to a pushbutton to allow the user to switch to another area of the application. CLOSE FORM may also be used to exit from a form.

Programmability: A Caution

Because this is a programmable feature, the form designer must be sure that the rules of programming are respected. In particular, if you call or exit to a subroutine in a DML program, you should avoid overly complex looping or branching structures that jump from one subroutine to another. You are advised to keep called subroutines simple and self-contained, and always to return to the form before calling another routine.

If your requirements are inherently complex, you should consider fully controlling the use of the form from within an independent DML program.

Check Boxes

The check box object type provides the user with a way of controlling the values entered into a field or variable.

A check box has two possible states: selected or unselected. Each state has a specific value attached, such as 'yes' or 'no,' 0 or 1. When the user turns the check box on, the value for the selected state is assigned to the field; when the check box is turned off, the value for the unselected state is assigned.

When used with variables, check boxes, like radio buttons, provide a way of indicating user choices, so that, for example, a DML program can perform selective processing based on those choices.

Defining a Check Box



You place a check box on a form by selecting the check box object type and clicking on the form.

1. Select the Check Box object type in the toolbox.
2. Click where you want the check box to appear.
3. Choose a field from the list shown in the field selection requester.
 - ☐ You may enter a variable manually.
4. Click OK.
5. In the Check Box Values requester, enter values for the selected and unselected states.
 - ☐ The data should match the data type of the field.
 - ☐ If a field has a validation formula attached, the values assigned to the field by a check box control must pass the validation.
6. Click OK.

The Form Designer places a check box object on the form, with a small dotted box extending to the right. This is the selection area for the check box. If you want to be able to turn the check box on and off by clicking the text that normally accompanies a check box, you should resize the selection area and type in the text in the usual way.

During data entry, a check box acts like a field, so it appears in the normal data entry order. You can click it to turn it on or off, or tab to it and use the SPACEBAR.

Editing a Check Box

1. Select the Check Box object type in the toolbox.
2. Click the check box to be edited.
3. Alter the values for the selected and unselected states.

You may edit the accompanying text for a check box in the normal way, but you must first resize the selection box so that it does not overlay the text object, preventing you from selecting it.

The text may appear in any color, font or text style.

Radio Buttons

A radio button is a type of form control. It allows you to attach a specific value to a field or variable, and assign that value by selecting the radio button.

If you define several radio buttons for a field, the buttons act as a group, allowing you to create effective *multiple choice fields* on a form.

Used with variables, radio buttons, like check boxes, provide a way of indicating user choices, so that, for example, a DML program can perform selective processing based on those choices.

Defining a Radio Button

1. Select the Radio Button object type from the toolbox.
2. Click where you want the radio button to appear.
3. Choose a field from the list shown in the field selection requester.
 - ☐ You may enter a variable manually.
4. Click OK.
5. In the Radio Button Value requester, enter the value to be assigned to the field when the radio button is selected.
 - ☐ The data should match the data type of the field.
 - ☐ If a field has a validation formula attached, the values assigned to the field by a radio button control must pass the validation.
6. Click OK.



The Form Designer places a radio button object on the form, with a small dotted box extending to the right. This is the selection area for the radio button. If you want to be able to turn the radio button on and off by clicking the text that normally accompanies a radio button, you should resize the selection area and type in the text in the usual way.

During data entry, a radio button acts like a field, so it appears in the normal data entry order. You can click it on or off, or tab to it and use the SPACEBAR.

If you want to be able to use the direction keys to move within a group of radio buttons, you must ensure that all the buttons in the group form a continuous sequence in the data entry order.

Editing a Radio Button

1. Select the Radio Button object type in the toolbox.
2. Click the radio button to be edited.
3. Alter the value that is assigned to the field.

You may edit the accompanying text for a radio button in the normal way, but you must first resize the selection box so that it does not overlay the text object, preventing you from selecting it.

The text may appear in any color, font or text style.

Using Form Controls in DML

Your application may call for forms that are opened and processed entirely within a DML program environment. The DML WAIT MOUSE command allows you to detect a mouse click on a pushbutton, radio button or check box. Radio buttons and check boxes must be attached to variables rather than fields before they can be used in this way.

A typical use for this command would be to wait for the user to make certain radio button selections and then click a pushbutton, which would direct control to the routine appropriate to the user's choices.

When a program receives a click on a pushbutton, the pushbutton command may direct control anywhere for example, to another routine within the same program (GOTO labelname), or to a new DML module (CHAIN programname).

10 GENERATING REPORTS

Superbase provides many alternative ways of obtaining output from database files. You can print single records, list file data record by record, or print forms that reproduce the graphics created with the Form Designer. For many users, though, the process of retrieving and outputting data is thought of as 'reporting.' In Superbase, reporting is considered as an extension of the program's query facilities, in which explanatory text, such as headings and footings, is combined with analytical functions, such as subtotalling and mean calculation, to produce more highly structured and informative output.

The Form Designer provides a set of commands for generating reports. You can use the easy point-and-click techniques of the Form Designer to position fields, calculations, report functions and text where you want them to appear in the actual report.

When you save the report form, the Form Designer generates a Superbase DML program and stores it on disk. This program may be run from within the database, and may also be edited to add further details if necessary. The report form itself may be edited in the Form Designer if you wish to change the report layout using editing techniques rather than programming.

Overview

The commands for creating reports are on the Report Generator menu. This menu is only enabled if the following conditions exist:

1. An existing report form has been opened,

OR

2. You have used the Project New command to create a new form **AND** you have opened a database file.

On the Report Generator menu, there are commands for defining headings and footings, commands for specifying group and global analysis features, and a command for selecting the fields to be printed.

Procedure Summary

When you create a report, you follow the same general procedure each time. Within this, of course, there is room for infinite variation, and you will need to understand the sections that follow this overview to appreciate the range of facilities available.

1. Enable the Report Generator menu as explained above.
2. Open all the database files you wish to use in the report.

3. Choose the Report Generator Select command to create a **SELECT** box for the body of the report.
4. Place fields in the **SELECT** box in the usual way, by selecting the **FLD** object type and clicking where you want the field to appear.
 - ☐ You may also place calculations (derived columns) and text in the **SELECT** box.
5. Choose the Report Generator Heading command to create a **HEADING** box in which page headings will appear.
6. Place text in the **HEADING** box in the usual way.
 - ☐ Calculations may be used to show date, time, or page number.
7. Repeat the last step using the Report Generator Footing command.
8. If you want your report to be grouped on the contents of a particular field, choose the Report Generator Group command to create a **BEFORE GROUP** and **AFTER GROUP** pair of boxes.
9. Specify text, fields, and report functions – the **FNC** object type – in the **BEFORE GROUP** and **AFTER GROUP** boxes. Subtotals are defined in the **AFTER GROUP** box.
10. Repeat steps 8 and 9 for each group field in a report.
11. For pre-report and post-report features like those available in the **BEFORE** and **AFTER GROUP** boxes, choose the Report Generator Global command to create **BEFORE REPORT** and **AFTER REPORT** boxes.
12. Use Define SB Links to establish a link between the form and its database files.
13. Save the report form.
 - ☐ The Form Designer creates both an **SBV** form file and an **SBP** Superbase DML program file.

When you want to run the report, exit from the Form Designer and run Superbase, then open the form in the usual way.

Example Report Form

HEADING				
SUPPLIERS LIST ON DATA				Page: 001
Code	Name	----- Last Paid ----- Date	Amount	Balance
BEFORE REPORT				
BEFORE GROUP SupState.SUPPLIER				
Suppliers for State: SupState.SUPPL				
SELECT				
Supplier	Supplier Name	SUPPLIER	Last Paid	Last Paid Current Ba
AFTER GROUP SupState.SUPPLIER				
Number of Suppliers:		Supplier Name	Balance Subtotal:	Current Ba
AFTER REPORT				
END OF REPORT				
Report Count:		Supplier Name	Report Total:	Current Ba
+				

Example Report Program

Saving the example report above would generate a program like this:

```

OPEN FILE "SUPPLIER"
REQUEST "REPORT TO PRINTER?", "", 1, a%
IF a% THEN PRINT ;
REPORT Current_Balnce.SUPPLIER
AFTER REPORT
? @31;"END OF REPORT"
?
? @9;"Report Count:";@23;&2.0 COUNT ;@50;"Report Total:";
@65; &10 SUM Current_Balnce.SUPPLIER
END REPORT
  
```

HEADING

```
? @26;"SUPPLIERS LIST ON";dat$;@66;"Page: ";pag$
?
? @1;"Code";@10;"Name";@44; "---- Last Paid ---- "
;@68;"Balance"
? @44;"Date";@57;"Amount"
?
END HEADING
```

```
GROUP SupState.SUPPLIER, Current_Balnce.SUPPLIER
BEFORE GROUP SupState.SUPPLIER
?
? @1;"Suppliers for State: ";@23;&2SupState.SUPPLIER
?
END GROUP
```

```
AFTER GROUP SupState.SUPPLIER
?
? @2;"Number of Suppliers: "; &2.0 COUNT ;@46; "Balance
Subtotal: "; @65; &10 SUM Current_Balnce.SUPPLIER
?
END GROUP
```

```
SELECT @2;&8Supplier_Code.SUPPLIER; @11;
&30Supplier_Name.SUPPLIER;@44;&8Last_Paid_Date.SUPPLIER
; @53; &10Last_Paid_Amt.SUPPLIER; Ince.SUPPLIER
NEWLINE
ORDER SupState.SUPPLIER ASK
```

Objects on Report Forms

The Form Designer allows only a subset of object types on the form: Text, Fields, Calculations, and Report Functions.

The report form is displayed in the system font.

Editing Report Forms

You can edit the boxes on a report form by selecting them and using the Size tools or the Clear command on the Edit menu.

Objects on report forms may be assigned underlined or italic text styles.

Sorting

A default sorting order for the report is generated automatically when necessary. The Form Designer derives this from the sequence of Groups in the report

structure, and builds an ORDER line, which is placed in the Superbase report program.

When you run the report program from within Superbase, you will be asked whether you wish to add any extra fields to the ORDER line. You might wish to sort within the records by date or amount, without actually specifying a group for such fields.

All the fields used for sorting the report are sorted in ascending order. If you wish to change this you can alter the ORDER line when it is presented at runtime.

See **Modifying a Report Program** at the end of this chapter.

Filter

A report that outputs fields from more than one file must have its files linked. The stored file links are used to generate a default WHERE statement in the Superbase DML report program that is created when you save the form.

The WHERE line contains only the information needed to relate files together when more than one file has been used in a report. The information is taken from the field links that you should define before you save the report form. If only one file has been used or no links have been defined, no WHERE line is created.

When you run the report program from within Superbase itself, the program displays a filter requester, so you can add extra conditions to the automatically generated filter line for use at runtime.

You can easily make these temporary extra conditions permanent by editing the report program and saving it, under a separate name if you prefer. See **Modifying a Report Program** at the end of this chapter.

Field Selection

The fields defined in the Select box will be printed on each line of the body of the report. In a report on a single file, the contents of the Select box will be fields from that file, and perhaps calculations used as derived columns.

Where the report uses fields from more than one file, the Select box represents the 'many' part of a one-to-many relationship. The 'one' part is represented by a Group box.

For example, suppose you wish to print a report of customer orders sorted by customer, and the data is held in two files, **Customer** and **Orders** (many Orders to one Customer). You would print details of each customer's orders as the body of the report, defining those fields here in the Select box. However, details about the customer would appear in a Group box which only prints out when it is time to change to the next customer.

Placing Fields in a Select Box

1. Choose the Select command from the Report Generator menu. The Form Designer creates a box across the screen with the keyword SELECT to identify it.
 - ☐ The box is three lines deep in the system font size.
2. Choose the Field (FLD) object type from the toolbox.
3. Click in the Select box where you want a field to appear.
4. Choose a field from the list of fieldnames. Click the one you want, then OK, and the field is placed in the Select box.
 - ☐ To change to a list of fields from another open file, click the arrow button next to the filename.
 - ☐ If you choose fields from more than one open file, you must link the files so that Superbase will know how to join them when the report is run. The file links are used to generate a default WHERE statement in the Superbase DML report program that is created when you save the form.
 - ☐ If you want to Sort the report output by group, you should NOT place the group fields in the Select box. Create Before Group and After Group boxes for this purpose with the Group command.

You may include text objects and calculations in the Select box, but not commands. To place some text in the Select box, select the Text object type in the toolbox, click in the Select box, and type the text in the usual way.

To place a calculation in the Select box, create one as described in Chapter 7 Form Calculation Formulas.

Sizing the Select Box

You can resize the Select box from its default three line depth. If you leave a blank line above or below the fields in the box, this will be reproduced in the report.

- By making the Select box very large, you can obtain page oriented output.

Heading

If you want a heading to print on each page you must define the text in a Heading box. To print a separate title page, use the Global command.

When you select the Heading command, the Form Designer creates a Heading box three lines deep in the system font. You may resize the Heading box to accommodate multiple lines and column headings.

Report Heading

To create a report heading, choose the Text object type in the toolbox, click where you want the text to appear, and type in the text. Report text may use any of the text styles.

Column Headings

You will probably need to resize the Heading box.

1. Select the Size tool in the toolbox.
2. Drag the bottom edge of the box downwards to make it larger.
3. Select the Text object type in the toolbox.
4. Click where you want the text to appear.
5. Type the text, pressing ENTER to finish.

Date, Time, and Page Number

You may include the system date and time and the page number in the Heading box. Create a calculation (see Chapter 7) and supply the system variables TODAY, NOW, and PG as appropriate. You may format the page number with the formula `STR$(PG,"99")`.

Footing

If you want a footing to print on each page you must define the text in a Footing box.

When you select the Footing command, the Form Designer creates a Footing box three lines deep in the system font size. You may resize the Footing box to accommodate multiple lines.

Follow the same procedure as for Headings when typing in the text for the Footing box or placing a calculation in it.

Sorting by Group

Report output must often be sorted into groups, such as lists of addresses by state or employees by department. The Group command allows you to select fields on which to sort the report. The fields you select are used to generate a default ORDER statement in the Superbase DML report program that is created when you save the form.

When you select the Group command, the Form Designer presents a requester from which you choose a field to group on. In some other programs this is called a 'break' or 'control break' field.

Click the field you want, changing file if necessary to display the fieldnames for each file, then click OK.

The Form Designer creates two boxes, one above and one below the Select box: the Before Group and After Group boxes. The two boxes are paired, and the name of the Group field is shown on the top line of each box to indicate this.

Multiple Groups

There is no practical limit to the number of groups you can create. Each pair of Group boxes is placed inside the previously created pair, so you should set up the groups in order from largest (e.g. Employee Company) to smallest (e.g. Employee section).

Before Group

If your report groups employee data by department, you should place fields to print details about the department in the Before Group box to avoid having to duplicate them in every line of the Select box.

The Before Group box contains text and data that you want to appear on every change of group. You can include field names that do not appear in the Select box. This is the way to produce reports that follow a transactional structure:

```
Customer number, customer name
    Order number, Order date, Order amount
    Order number, Order date, Order amount

Customer number, customer name
    Order number, Order date, Order amount
    Order number, Order date, Order amount

Customer number, customer name
    Order number, Order date, Order amount
    Order number, Order date, Order amount
    Order number, Order date, Order amount
```

Here, the Customer data is placed in a Before Group box for the Customer name or account number, and the Transaction data appears as a group of fields in the Select box.

Subtotals and Other Report Functions

The After Group box may contain text and fields that you want to appear when a group ends. This is where you analyze the report data with functions for totaling, averaging, etc.

To add a Report Function, choose the Function object type from the toolbox and click where you want the function output to appear. For example, you would

probably want to align a subtotal result with the field in the Select box to which it applied.

You may, however, position Report Functions anywhere within an After Group or After Report box, as they always specify the field to which they apply. You will probably want to add some text next to the function to identify its purpose.

1. Select the Report function object type (FNC) from the toolbox.
2. Click where you want the results of the function to appear.
3. When the Report function requester appears, click one of the keyword buttons, then click the field name to which it is to apply:

SUM Balance.CLIENTS

4. Click OK to complete the definition.
 - You can place only one fieldname after the function keyword (only one result can appear where you clicked), so repeat this procedure for each function.

In report forms, Report functions may be placed only in the After Group and After Report boxes.

Report Function Syntax

The following functions may be used in Reports.

SUM (field) Gives the field total for all the records in a group or report.

MIN (field) Returns the minimum value of the specified field among the records in a group or the report.

COUNT Returns the number of records in a group or the report.

MAX (field) Returns the maximum value of the specified field among the records in a group or for the report.

MEAN (field) Returns the average value of the field data in a group or in the whole report.

VAR (field) Provides the Variance measure of the spread of the data from its mean.

SD (field) Provides the Standard Deviation measure of the spread of the data from its mean for the field data in a group or over the entire report.

- Apart from COUNT, the report functions may only be used with numeric fields.
- You may combine report functions with each other and with other functions to derive more complex reporting functions.
- If you want to force a new page after a group, choose Page on Group in the Report Generator Options requester.

- When a group keyword changes, the group field no longer contains the data for the old group. For example, when grouping on the field **State** and the state changes from 'Alabama' to 'Alaska' the value 'Alabama' is no longer accessible in State. However, you may use the keyword **GROUP** in a calculation to retrieve the old value for use in an After Group box.

Global

If you wish to print grand totals for a report at the end, you need to use a separate box to define the text and functions you wish to see.

Choose the **Global** command. The Form Designer creates a pair of boxes, the Before Report and After Report boxes.

Before Report

Add any text that you want to print as a once only title for the report.

If you want the title page to be printed separately, choose **Page before Report** in the Report Generator Options requester.

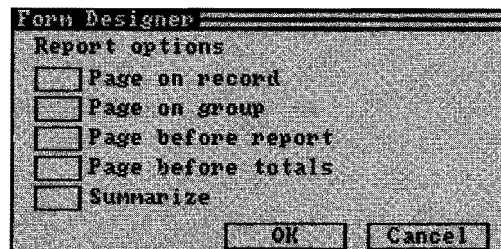
After Report

To create totals, etc., use the techniques described above in Subtotals and Other Report Functions.

If you want the totals page to be printed separately, choose **Page before Totals** in the Report Generator Options requester.

Options

The **Options** command on the Report Generator menu allows you to specify paging and summarization details.



Page on Record

You may want to force your report to start a new page for each record. If so, click this item so that the box is checked before you click OK.

Page on Group

You may want to force your report to change to a new page at the end of certain groups. For example, you might want to print the details of employees in each department separately.

When you click this item, the Form Designer shows a list of existing groups. Select a group and click OK.

The Form Designer places the word EJECT in the top line of the relevant AFTER GROUP box. This command works as a toggle, so to switch it off go through the actions again for the same group.

Page before Report

If you want to print a separate title page, define the title text in a Before Report box, and click this option.

Page before Totals

If you want to print a separate totals page, define the contents of the After Report box, and click this option.

Summarize

If you want your report to omit the details in the Select box, usually in order to obtain the results of After Group or After Report without the full print out, select this option.

It may be convenient for you to save two versions of the same report under different names, one with and one without the Summarize options selected.

Test

The Test command on the Report Generator menu enables you to test a report form on the printer before you save it. Selecting this command gives you a preview of what the output will look like when the report form is used in Superbase. If you find that is not as you planned, you can then go back and edit the form.

The preview shows how the report form will structure and lay out fields and text over one page. It restricts itself to six lines of repeated fields, and uses the Field character (as set in the Print Options requester) in place of the field data. When the report is output from Superbase, each page may contain data from many more than six records, so the preview may not look exactly the same; but it will show whether the fields and text are positioned correctly.

Saving the Report Form

When you have specified all the elements of the report form, save it in the usual way, using Project Save. The Form Designer generates the two files that the system will need to allow you to execute the report or load it again into the Form Designer for modification.

- A new SBP report program is generated every time you save the SBV form file. If you have modified the program code with the Superbase Program Editor, you should rename the program if you do not want it to be overwritten by the newly generated version.

Formatting Report Output

Calculations are especially useful for formatting text and numbers in the report output. Thus to format a numeric field, you can use the function STR\$. The field should be included in a calculation formula instead of being added to the form as a separate object. For example, the calculation **Balance\$** could be used to format the numeric field **Balance**, using a formula like this:

`STR$(Balance, "$999999.99")`

Another example, which shows how text can be formatted, would be a formula such as:

`Firstname + " " + Lastname`

Here the formula ensures that first and last names are correctly spaced in the report output.

Running a Report

The Form Designer is used to create a report form. To run the report, you need to return to Superbase.

Report forms are different from other forms. When you open a report form from within Superbase what you see is not the form but its output, either on the screen or the printer. This is because a report form is only a way of generating a report program. Opening the report form causes Superbase to execute the program.

There are two ways of running a report: opening the report form, or executing the report program.

If you simply want the report to run without any further actions, choose the Project Open Form command, and then select the report you want. You are offered the option to run the report program automatically.

The other choice is to treat the report like any other program. Select the DML Open command, then the name of the report program you want. It will be in the list of Superbase programs in the current directory.

Once the report program is loaded, you can either run it immediately, or use the program editor to examine or modify the report program code. To do the latter, select DML Edit.

Adding Filter Conditions

The Form Designer generates a WHERE line in the report program which contains the default relational information derived from the report form's internal structure.

The default relational information is only generated if more than one file is used in the report form. It always takes the form:

WHERE Field1.FILEA = Field2.FILEB ASK

where the fields contain the information on which a join can be made, such as an account number held in both files.

Access to the default filter is automatically provided when you run the report program by the ASK statement at the end of the line.

When Superbase executes this instruction, it displays the standard Query Filter Definition requester, as obtainable from the Process Query menu command.

You may either click on OK, if you wish to make no changes, or add to the Filter Command Line by clicking on fields and entering values and functions to build up a more complex set of search conditions.

It would be unwise to remove the relational join, as Superbase may begin outputting extremely repetitive and confusing data, and continue for a very long time (the process won't harm your data).

Click OK to call up the Order requester.

Adding Order Conditions

The report generator derives a default sorting order from the groups in the report form.

When you run the report, Superbase executes an ORDER line which includes the command ASK. This causes the presentation of the Query Order requester.

You may add extra fields to the Order line, or change sorting order from Ascending to Descending for any particular field.

Click OK to confirm your selections.

Print/Display Option

Before beginning output, the report program executes an instruction to select Screen or Printer as the output device.

Indicate your preference and click on OK. The report program executes the rest of its instructions.

Modifying a Report Program

It is quite likely that you will want to make some changes to the way your report program executes. The main areas where changes are required are:

- Specifying a fixed output device
- Specifying fixed or extra default filter conditions
- Specifying fixed or extra default order conditions

Output Device

The output device is selected with the line:

```
REQUEST "REPORT TO PRINTER?", "", 1, A% :  
IF A% THEN PRINT;
```

To fix the output device as the printer, remove this line, and add the statement TO PRINTER to the end of the report program (before the END SELECT statement if there is one).

To fix the output device as the screen, remove the whole line: the default output device is the screen.

To fix the output device as a disk file with the name 'tempfile,' remove the line as above, and add TO FILE "tempfile" to the end of the report program (before the END SELECT statement if there is one).

Filter and Order

You may want to extend the range of the default commands, while keeping the ability to add different parameters each time you run the report.

Edit the report program. Add any conditions you want to the WHERE line, removing the ASK command only if you don't want the command to add conditions at runtime.

Do the same for the ORDER line to change or extend the sorting parameters for the report.

11 PRINTING FORMS

The Form Designer allows you to print forms for two reasons: as part of the design process, and to serve as masters for reproduction.

See Chapter 12 Housekeeping for details of how to print out a report of the contents of a form.

Printer Setup

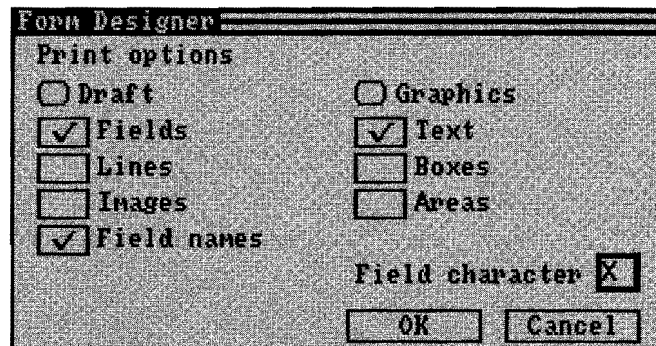
When you want to select a printer, use the Printer Setup command to list the printers currently installed for your system. Click the one you want, then OK.

Windows then displays the Page Setup requester for the selected printer. This allows you to see the current settings for page size, margin sizes and so on, and to change them if necessary. If the selected printer offers a choice of typefaces and/or sizes, this requester allows you to cycle through the list of available typefaces until the one you want is displayed. When you are satisfied with the contents of all the fields in the requester, click on OK.

Printing Forms

You can print out a copy of the form with the Print command on the Project menu. This allows you to check the physical appearance of the form.

The Print Options requester allows you to choose the types of object to be included in the printout.



1. Select the Print command on the Project menu.
2. Click either Draft or Graphics. Draft is quicker but omits any graphical features that you may have included on the form – it uses the current Superbase printer control file to output the form's text and the contents of the fields. Graphics performs a graphics screen print of the whole form.
3. Check the object types that you wish to print out.

4. Click the Field Names option to determine whether field names appear within the fields when they print.
 - ☐ The Field Character option determines which character is used to 'fill' the extent of the field and show its full dimensions on the page.
5. Click OK.

12 HOUSEKEEPING

Housekeeping covers a variety of different topics: looking at the disk directory, changing the directory, removing forms from disk, and removing pages.

Directories and Subdirectories

It is a good idea to store forms and pages in a separate directory from the program itself. You may want to create several subdirectories; for example, one for Superbase files and one for forms and pages. But if you intend to use a form for displaying Superbase data, it should be in the same directory as the Superbase file.

Your AmigaDOS manual tells you how to create a subdirectory. This needs to be done before you load the Form Designer.

Listing the Directory

The Project Directory List command lists all the files in the current directory. When you select it from the menu, a requester appears giving you the choice of outputting the directory list to the printer or to the screen.

Status

The Status command on the Project menu provides two types of information. First it gives information about various aspects of the system. Then, if there is a form in memory, it displays the details of the form's fields and calculation.

You may examine or print out details of form structures. This is an important part of setting up an application, and you should keep form status reports together with file status reports printed from the database.

When you select Status, a requester appears giving you the choice of outputting to the printer, to the screen or to a file. Select the destination you want, then click OK.

The Project Status command shows the following general information:

- Form Name
- Directory
- Disk Space Free
- Memory Free
- Number of Transactions in the Form
- Number of the current page
- A list of the files used in the form
- The current inter-file Link Structure

Superbase Form Designer

File: MGR:abdata/stks page 1

Status for form: MGR:abdata/stks

Directory: MGR:abdata

Disk space left: 92824342

Form: 632416

Control codes: 627836

Transactions: 10

Pages in form: 6

Open files: CLIENTS TRANS COUNTRY

File links: CURRENCY STOCKS CURRENCY CURRENCY COUNTRY STOCKS

Stock Rev: TRANS COUNTRY STOCKS

Customer: CLIENTS COUNTRY STOCKS

Name: COUNTRY

Page 1

TYPE	NAME	ATTRIBUTES
CLC	SELECT FORM KEY selkey	1

Page 2

TYPE	NAME	ATTRIBUTES
FLD	STOCK	1
FLD	STOCK	2
FLD	STOCK	3
FLD	STOCK	4
FLD	STOCK	5
FLD	STOCK	6
FLD	STOCK	7
FLD	STOCK	8
FLD	STOCK	9
FLD	STOCK	10
FLD	STOCK	11
FLD	STOCK	12
FLD	STOCK	13
FLD	STOCK	14
FLD	STOCK	15
FLD	STOCK	16
FLD	STOCK	17
FLD	STOCK	18
FLD	STOCK	19
FLD	STOCK	20
FLD	STOCK	21
FLD	STOCK	22
FLD	STOCK	23
FLD	STOCK	24
FLD	STOCK	25

SPRCE: continue CTRL-C: quit

Below this are listed details for every field, calculation and command on the form.

Below this (if there is a form in memory) Status gives the details of the form's field and calculation objects. For each object it shows the type (either FLD for field or CLC for calculation), the name, and the object's attributes.

The attributes are listed in four columns. From left to right, they are:

Order. The object's order entry number.

Justification. This can be L, C, or R for left, centre and right justification.

Read Only. If the object has been given the read only attribute, it is indicated by the letter R. Otherwise this column is left blank.

Display Only Status. If the object is printable, this is indicated by the letter P.

If a calculation has a formula attached to it, this is shown in the line below.

Removing Forms and Pages

You can delete an SBV file from disk with the Remove command in the Project menu. In the case of a Report Form which has an SBP program file of the same name, the program file must be deleted from within the database.

This command deletes the specified file from disk, and makes the space it occupied available for further storage. Use Project Remove when you are running short of disk space, or simply to keep your directories tidy by deleting files that are no longer used.

Remove PG form page files with the Remove command in the Page menu.

13 COMMAND REFERENCE SUMMARY

Project Menu

New

Use this command to clear the work area and start a new form.

Open Form

Displays a list of all SBV form files in the current directory.

Open SB File

Allows the opening of a database file definition for selection of fields for use on a form.

Open dBase File

Allows the opening of a dBase file for selection of fields for use on a form.

Save

Saves the current form file.

Save As

Saves the current form file under a new name

Remove

Deletes a form file from disk.

SB File Close

Closing files normally releases some system memory, which can be helpful when more than one application is contending for available memory space.

Directory List

This command lists the contents of the current directory.

Status

You may examine or print out details of form structures. This is an important part of setting up an application, and you should keep form status reports together with file status reports printed from the database.

Printer Setup

Allows you to select a printer from those available, then to set page and margin dimensions and select a printer typeface.

Print

The Print Options requester allows you to choose the types of object to be included in the printout. Check the types you wish to see.

About

Identifies the version of the Form Designer in use.

Quit

The Quit command leaves the application and returns either to Superbase or to the Workbench, depending on whether the Form Designer was called from the database or the Workbench.

Edit Menu**Undo**

The Undo command reverses the previous editing action. Actions that may be undone include Move and Size. Undo also removes the last object placed on a form if your last action was to create it.

Cut

Removes the selected object and stores a temporary copy.

Copy

Stores a temporary copy of the selected object.

Paste

Inserts a copy of the stored object or objects.

Clear

Removes the selected object from the form.

Tools

Hides or displays the toolbox.

Snap to Grid

You may want objects to be automatically aligned to the current grid. When the Edit Snap to Grid command is selected, objects snap to the grid when you create or move them.

Ruler/Grid

The Rulers and Grid editing aids are available to assist you in aligning objects on the form.

Crosshairs

When the Crosshairs command is selected, vertical and horizontal lines are shown intersecting at the mouse pointer. The intersection is shown as pixel co-ordinates to the right of the toolbox.

Auto Field Names

When the Auto Field Names option is selected, the name of any field that you place on the form is positioned next to it.

Reduced

If your form is larger than the window, you may wish to see it scaled down to check that the parts are correctly arranged. Select the Reduced View command to display a scaled down view of the form.

Page Menu**New**

When you want to create a second or subsequent page for your form, use this command.

Open

Lists all the PG files in the current directory so you can choose one to insert into the current form.

Save As

Allows you to save the current page of the form as a PG file.

Remove

Lists the PG files in the current directory so you can select one to delete from disk.

Setup

The Form Designer allows you to create forms of practically any size supported by your printer. The Setup command lets you adjust the size of the form's work area to that specified for the printer.

Hierarchy

Allows you to alter the existing hierarchy of objects on a form.

Clear

If you want to remove all objects from a page, use the Page Clear command.

Erase

In a multi-page form, you may need to remove a page completely. The Page Erase command deletes the current page.

Next

Move to the next page of the form that is being designed.

Previous

Move to the previous page of the form that is being designed.

Goto

Move to a specified page of the form that is being designed.

Define Menu

Font

Text and field data may be rendered in any font available to the system.

SB File Link

When you design a form with fields from more than one file on it, you must use the Define SB File Links command to set up the information that Superbase will need to process the form correctly.

Data Entry Order

You can define the sequence to be followed when you enter data into a form. This allows you to follow logical sequences without being tied to an inflexible structure. You may even jump between different pages of a form if you wish.

Transaction Lines

Many applications include one-to-many relationships between records in different files. In a form, you can define a 'block' of fields as the 'many' part of such a relationship. The term 'transaction lines' refers to such a block.

Color Palette

The Color Palette command allows you to redefine the colors used in the toolbox.

Resolution

The default number of colors available under Workbench 1.3 is four. More colors are obtainable by changing the resolution using the Define Resolution command.

Report Generator Menu

Heading

If you want a heading to print on each page you must define the text in a Heading box. To print a separate title page, use the Global command.

Select

The fields defined in the Select box will be printed on each line of the body of the report. The Select box represents the 'many' part of a one-to-many relationship. The 'one' part is represented by a Group box.

Footing

If you want a footing to print on each page you must define the text in a Footing box.

Group

When you select the Group command, the Form Designer presents a requester from which you choose a field to group on. In some other programs this is called a 'break' or 'control break' field.

Global

If you wish to print grand totals for a report at the end, you need to use a separate box to define the text and functions you wish to see.

Options

The Options command on the Report Generator menu allows you to specify paging and summarization details.

Test

Especially with wide reports, it may be desirable to obtain an example of dummy printout before ending the report design.

Toolbox

Areas

An area is a rectangle on a form. It may exceed the window size, and be drawn in any color or pattern available. A bordered area tool type is also available. The border color of a bordered area is set separately from foreground or background colors in the toolbox. Areas may be square or round cornered.

Boxes

A box is a rectangular area on a form. It may exceed the window size, and be drawn in any color or line thickness available. Boxes may be square or round cornered.

Lines

Lines are useful for dividing boxes or areas of the screen either horizontally or vertically.

Text

You may wish to place text on a form for various reasons: to act as a prompt to the operator, to clearly identify the nature of the application, or to mark out groups of fields more clearly.

Images

You may wish to place images on a form to improve the design or to allow the printing of forms with logos or special lettering. You might wish to use small images such as arrows or symbols to make the form easier to use.

Check boxes

The check box object type provides the user with a way of controlling the values entered into a field or variable. A check box has two possible states: selected or unselected. Each state has a specific value attached, such as 'yes' or 'no,' 0 or 1. When the user turns the check box on, the value for the selected state is assigned to the field; when the check box is turned off, the value for the unselected state is assigned.

Radio buttons

A radio button is a type of form control. It allows you to attach a specific value to a field or variable, and assign that value by selecting the radio button. If you define several radio buttons for a field, the buttons act as a group, allowing you to create effective *multiple choice fields* on a form.

Move Tool

Select the four-way pointer tool in the toolbox. Now you can click an object and drag it to a new position on the form in one movement.

Fields (FLD)

Placing a field on a form allows you to enter data into database records and retrieve it with the same form. You can select fields from any existing database file, and position them anywhere on a form.

Form Calculations (CLC)

Calculations are formulas that may be added to a form to extend its usefulness. You may use calculations to compute and display results from data entered elsewhere on the form or to transfer form data to fields.

Validation (VAL)

If you are dealing with important information, you need to be confident that stored data is accurate. The only time you can control the quality of your data is when it is first entered. Validation is a way of ensuring that the data entered into a field meets certain criteria.

Commands (CMD)

By placing commands on a form, you can greatly extend its ability to process data as it is entered. A command is one or more DML statements entered as a single line. The command must follow DML syntax as defined in the DML Reference Guide.

Report Functions (FNC)

Report functions provide a means of using statistical features such as MEAN and SUM in both report and ordinary forms. Report functions have a special keyword syntax.

Size Tool

Objects on forms frequently need to be resized. This tool allows you to vary the size of any type of object except text.

Colors

The default number of colors available under Workbench 1.3 is four. More colors are obtainable by changing the resolution using the Define Resolution command.

Color Selector

The Color Selector (to the right of the Move tool) shows the current colors, in which objects and text will be rendered. There are two areas in the Color Selector, a center and a surround. These represent the pen and paper colors respectively.

Border Color Selector

To the right of the main Color Selector is the Border Color Selector. The color shown here determines the color in which the borders of areas and fields are rendered.

Pen and Paper

(Below the left half of the main Color Selector.) When a text object, a field, a calculation, or a command is created, it has both a pen color, in which the characters are drawn, and a paper color, which is used for the background to the character. Areas which are drawn in a pattern of dots also have a foreground color and a background color.

Field and Area Borders

This tool (below the right half of the main Color Selector) places a border round each field or a selected area.

Rounded Corners

By default, boxes and areas are drawn with square corners. When you select this tool, they are drawn with rounded corners.

Text Style

Text objects and field data may appear in plain, boldface, italic or underlined text styles.

Currency Calculation

The dollar tool is used to designate form calculations as currency type. When you create a numeric calculation with this tool selected, its results will be shown to two decimal places, right aligned.

Read Only

You may wish to protect certain fields from being edited by users. This tool allows to designate fields, calculations, and commands as read only, which means that Superbase will never allow them to be changed.

Field Justification

Fields may be justified left, center, or right. Click the selector to show the required position in the Justification Selector.

Display Only

You can designate any object on the form as display only. This means that it will not be printed when you print forms and data from the database.

Area Patterns

You can vary the appearance and the color of an area by drawing it in a pattern of dots. The default pattern is solid.

Line Patterns

Different types of line thickness may be used to differentiate different kinds of box, area, and line.

14 INTRODUCTION TO DML

Welcome to Superbase's Database Management Language (DML). DML is based on the programming language Basic. It includes most of the standard Basic commands and functions, but supplements them with a large number of commands and functions that are specific to database management. The database commands duplicate the controls that Superbase provides through its menus and requesters. This means that almost all of Superbase's menu-driven file and record handling facilities are available under program control.

Once you have familiarized yourself with Superbase's controls, the corresponding program commands will be easy to understand. If you haven't programmed before, you may find the idea of learning a program language daunting. But as far as DML is concerned, a little goes a long way, and you do not need to be fully conversant with the language in order to take advantage of it.

In effect, you are already following a program sequence every time you perform a task which involves a series of menu operations. Writing a program that performs the task for you is simply a matter of entering commands in the same sequence. Generally, you will be able to find a single command to duplicate each of the menu operations.

As you acquire more expertise, you can move on, building bigger and more complex programs by combining routines, until you have fully automated your database system. When you're ready, you can incorporate Superbase forms into your programs, taking advantage of their built-in facilities for generating and retrieving records in several files at once.

At the highest level, you can specify your own pull-down menus, replacing the standard Superbase menus with the options that are relevant to the job in hand. And you can customize your application to an even more detailed level by creating your own pop-up selection panels to guide the user's choices.

Using this Guide

Before reading Chapters 15 to 17 which comprise the DML User Guide, you will need to be familiar with Superbase's menu and keyboard controls. Many of DML's commands provide a program equivalent of a menu or keyboard option, and the descriptions given here presume that you already know how to use the corresponding option.

However, once you have mastered Superbase itself, you do not need to read these chapters all the way through. As a reference guide, they can be consulted as and when they are needed.

How to use this Guide

Chapter 15, **Overview**, describes the most basic features of DML. You can think of it as a grammar book for the language. It sorts DML's 'keywords' and other components into different groups and, more importantly, sets out all the rules for using them. For example, it specifies the maximum length for a program line, and gives the rules for creating variable names.

Chapter 16 explains how to use the Program Editor. This covers entering and editing programs, storing them on disk and loading them from disk. It also explains how to execute programs, and how to create a 'start up' program which will be executed automatically when you start a session with Superbase.

Chapter 17, **Keyword Reference Guide**, forms the bulk of the manual. In this chapter, you will find an entry for each DML keyword, which gives the keyword's syntax and describes its usage. The entries are in alphabetical order starting with the ? command. At the front of Chapter 17, there is an explanation of the conventions used to show a keyword's syntax.

Chapter 18, **DML Summary**, gives the syntax for each DML keyword together with a brief description of its function.

15 DML OVERVIEW

Operating Modes

DML has two modes of operation: direct mode and program mode.

Direct Mode

In this mode, DML executes instructions as soon as you have typed them in. First you need to select the Command option from the DML menu. Then enter your instructions – a single command or a line of commands separated by colons – in the command line window; when you select 'OK' or press the ENTER key, DML will carry out the instructions straight away. The command line window allows you to enter up to 255 characters and move within the window using the cursor keys.

Program Mode

In program mode, DML does not execute commands as you enter them. Instead they are stored in memory and executed only when the program is run. The main difference between this mode and direct mode is that with the latter you can only enter and execute one line at time; program mode allows you to enter a series of instruction lines which are carried out in sequence. Program lines can be up to 255 characters long.

Other DML Applications

DML's functions, along with its operators and variables, can also be used in other Superbase operations – such as field definition validations and calculations, filter conditions, update commands and query derived field definitions.

Keywords and Reserved Words

Any word that DML recognizes as a specific instruction, or part of an instruction, is known as a keyword.

A keyword cannot be used as a variable name, a field name, or a program label. In this context, keywords are also known as reserved words – DML reserves them for its own use, and will interpret them as such even if they are in lower case.

A reserved word can, however, form part of a name. For example, you can incorporate the reserved word TO in any of the following ways:

TOP:	(in a label)
TOTAL%	(a numeric variable)
tot\$	(in a string variable)
TOTALS	(in a field name)

But you cannot use it like this:

TO:
TO%
TO\$

Commands and Statements

Some programming manuals make a strict division between two kinds of executable instructions – commands and statements. Commands are those instructions which are generally executed in direct mode, while statements are instructions that can only appear in a program line.

In DML there are only a few instructions that cannot be used in both operating modes and so the two terms are used almost interchangeably. Thus we refer to a line with more than one instruction on it as a multi-statement line; but it could equally well be called a multi-command line.

Variables

There are three types of variables in DML: string variables, numeric variables, and arrays. In many circumstances, the fields in a Superbase file can also be treated in the same way as variables.

Variable Names

Numeric variables must end with the '%' character (frequently preceded by another special character, as described below), string variables end with a '\$' character. Apart from this, the names for both types of variable follow the same rules:

Variable names have a maximum length of 14 characters and a minimum length of 1 character (excluding '%' or '\$').

The first character of the name must be a letter in the range 'a' to 'z' but the remaining characters can be either alphabetic or numeric. DML is not case sensitive; 'abc%' is the same as 'ABC%', and 'abc\$' is the same as 'ABC\$'.

A variable name can contain a reserved word, but cannot consist of just a reserved word and the variable suffix, % or \$. For example, names like 'left\$', 'cos%', 'report\$' and 'FILE%' are forbidden.

Numeric Variables

Numeric values may be stored in four types of variable: integer, long, real and auto, with the type of a numeric variable being denoted by the final one or two characters of its name. An 'auto' numeric variable can hold any value that the other three types can hold, but efficiency of storage space and speed of execution is improved by the use of integer, long and real numeric variable types.

DML's numeric variables hold numbers at up to 14 figure accuracy, but if displayed or printed, they are shown in the current default numeric format.

The values that can be stored in the various numeric variable types, and their identifying characters, are as follows:

- *nvar%%* holds integer values from -32768 to +32767
- *nvar&%* holds integer values from -2147483648 to +2147483647
- *nvar#%* or *nvar!%* hold real values from 10e-307 to 10e+308
- *nvar%* is an auto numeric variable and can hold any of the above values.

If you assign a non-integer value to an integer or long field (or variable), Superbase will strip off the decimal part of the number and store just the integer part.

String Variables

String variables are used to hold ASCII characters – usually alphanumeric text. The maximum length of data that can be held in a string variable is 4000 characters.

Arrays

DML supports string and numeric arrays with up to three dimensions. The maximum number of elements in an array is limited only by the amount of memory available.

The rules for array names are the same as those for numeric and string variables. Numeric arrays must end with the '%' character, string arrays must end with the '\$' character.

System Variables

These are variables which are assigned a value by DML. Unlike user-defined variables, the names of the system variables do not end with the '%' or '\$,' and, with a few exceptions, they may be not be assigned a value in a DML statement. But otherwise they can be used in the same way as standard variables.

The system variables are: ERRNO, FILE, FORM, GROUP, INDEX, NOW, ORDER, POSITION, PG, TODAY, USERNAME, WHERE. Apart from TODAY and NOW, which give the system date and system time, they hold either numeric or string values. TODAY and NOW are numeric variables but DML treats them in the same way as date and time fields: it displays their values in the current date or time format. If the time of day is 9:41 and the system clock has been set to this

time, NOW would display it as 9:41. However, it stores this time as 34860000 – the number of thousandths of a second from midnight to 9:41.

You can assign a value to TODAY and NOW using the commands SET TODAY and SET NOW.

File Types

When you save a file using one of the SAVE options, DML automatically adds one of the following extension names to the file name:

- aaa.sbf is a database file
- aaa.sbd is a file definition for a database file
- aaa.dbd is a file definition for a dBase data file
- aaa.sbk is a function key file
- aaa.sbb is a label format file
- aaa.sbp is a program file
- aaa.s bq is a query file
- aaa.sbt is a text file saved from the text editor
- aaa.sbu is an update file
- aaa.### is an index (where ### is a number from 001 to 999)

DML stores program files, Text Editor files, database files and index files in its own format (for example, it tokenizes keywords in program files and stores formatting information with documents saved in the Text Editor). The others are all text files and only contain ASCII characters.

Fields

With a few exceptions, you can use fields in the same way as variables. For instance, you can input data directly to fields; you can supply a field name as the argument for a function; and you can assign values to fields.

Fieldnames must conform to the rules for field names as set out in Volume 1, i.e.:

The maximum length is 15 characters.

The minimum length is one character.

They must begin with an alphabetic character, but can thereafter contain any alphanumeric characters. They can also contain the underscore and space characters. Superbase will accept a field name with several spaces embedded in it – i.e., 'a bc d' is a valid name – but this is not recommended.

DML is not case sensitive; 'abc' is the same as 'ABC'.

The file a field belongs to must be open; otherwise the following error message appears:

'field name'
Cannot find this field

Where more than one file is open and there are two files with the same field name, the file name must be given as an extension to the field name and the two must be separated by a period. This enables DML to identify the field you are referring to.

Note Failing to supply file extension names to fields may result in incorrect data, although it does not cause an error.

If the file name contains spaces, it must be included within quotation marks, e.g.:

fielda.filea
name.Customer
Lastname."Customer file"

Date and Time

Superbase stores dates as julian date numbers; that is, as the number of days from 31st December in the year zero. This means that date fields are numeric fields and hold their dates as numbers. When you display a date field, however, Superbase automatically converts it to a more recognizable format: it takes the format which has been set in the file definition (see Chapter 3, Volume 1).

The date format only controls the way dates are displayed. You can enter a date from the keyboard in almost any format. For instance, the date might be shown on screen as 2/21/87, but you can type in a date like 'February 21, 1987' and DML will accept it.

Incidentally, DML does remember that eleven days were lost when the Gregorian Calendar reform was implemented (September 2nd, 1752 is followed by September 14th).

As with dates, DML stores the time in time fields as a numeric value, but displays it in a different format. The time is stored as the number of thousandths of a second from midnight. When you display the time, it is given in hours and minutes in either 12 or 24 hour format.

Functions

Functions make up an important group of DML keywords. Most of them perform a calculation on a number or a string, but there are also functions that give information about some aspect of the system. RECCOUNT, for example, returns the number of records in a file, and FREE returns the amount of free memory space.

Every function returns a value – either a numeric value or a text string. It follows that, unlike a command, a function cannot be entered on its own as a statement. Thus:

```
? RECCOUNT ("address")
```

and

```
Numrecs% = RECCOUNT ("address")
```

are valid statements, but

```
RECCOUNT ("address")
```

produces an error message.

All functions take at least one argument, which is enclosed in parentheses, and return a result. In the example above, "address" is the argument. It tells the function which file to count.

Depending on the function, an argument may be a field name, a file name, a constant, a variable, or a combination of these connected by an operator to form an expression. You will find details of what kind of arguments a function expects in the Keyword Reference section.

Functions are usually classed as either string or numeric functions, according to the type of result they return (which is not necessarily the same as the type of arguments they require).

To find out what type of result a function returns, there is a simple rule you can apply. Functions which have a \$ character at the end, return their results as string data; if they do not end in a \$, they return a numeric result. There is one exception to this rule – REPLICATE, which returns a string result, despite the fact that it does not have a \$ at the end.

A string function should not contain more than one intermediate string concatenation. For example:

```
LEFT$("abcde",LEN("x" + "y"))
```

contains only one intermediate string concatenation and gives the correct result, but:

```
LEFT$("ab" + "cde",LEN("x" + "y"))
```

gives the wrong result because it contains two concatenations – "ab" + "cde" and LEN("x" + "y"). The correct way to do this would be to concatenate "ab" and "cde" beforehand by assigning them to a string variable. For example:

```
a$ = "ab" + "cde"  
b$ = LEFT$(a$,LEN("x" + "y"))
```

Report Functions

These functions can only be used with repeated fields occurring in reports and transactions (a group of repeated fields in a form), and array variables.

When a report or a form displays multiple instances of the same type of data – for example, the deposits made to a bank in a single month – you will usually want to derive totals from the data or to analyze it in certain ways. Report functions allow you to do this with a single command. Within a program you can use them to total or analyze a range of values which have previously been assigned to an array.

The report functions are:

```
SUM  
COUNT  
MIN  
MAX  
MEAN  
VAR  
SD
```

Apart from COUNT, which returns the number of instances of the same field or the number of array elements, these functions only operate on numeric values: the function's argument must be a numeric field or a numeric array.

There is an important difference between the way the report functions work with arrays and with the fields in a transaction form. With arrays, they act on all the elements whether they have been assigned a value or not. But with transaction forms they only operate on the lines which contain data; i.e., COUNT shows the number of transaction lines that are filled and ignores blank lines.

Parentheses are optional with report functions. To show the total of all the occurrences in a transaction form of the field **Amount**, you could enter:

```
? SUM Amount
```

or

```
? SUM (Amount)
```

With array variables, you enter the array name on its own, as in:

```
? MEAN a#%
```

If the array has more than one dimension, the function operates on the first dimension.

Operators

DML provides three types of operators: arithmetic, relational and string.

Arithmetic Operators

Addition

The plus sign specifies that the operand on the right is added to the operand on the left, e.g.:

```
2 + 2
a#% + b#% + c#%
a#% + 12.225
```

Subtraction

The minus sign specifies that the operand on the right will be subtracted from the operand on the left e.g.:

```
3 - 2
a#% - b#% - c#%
a#% - 12.225
```

The minus sign can also be used to set an operand as a negative number. The effect is to subtract the operand from zero, e.g.:

```
-3
-a#%
```

Multiplication

The asterisk is recognized by DML as the multiplication symbol and specifies that the operand on the left is multiplied by the operand on the right, e.g.:

```
3 * 2
a#% * b#%
a#% * 12.225
```

Division

The slash is recognized by DML as the division symbol and specifies that the operand on the left is divided by the operand on the right, e.g.:

```
3 / 2
a#% / b#%
```

Note

Unlike in many other programming languages, division by zero does not cause an error message and returns a value of zero. This, of course, is a false result, and you should check numerical data to ensure it does not occur.

Exponentiation

The up arrow (or caret) is recognized by DML as the exponentiation symbol and specifies that the operand on the left is raised to the power specified by the operand on the right (the exponent); that is, the operand on the left is multiplied by itself the number of times specified by the exponent. If the exponent is 2, the number is squared; if the exponent is 3, the number is cubed; if the exponent is 1/2, the square root of the number results, e.g.:

$3 \wedge 2$	(3 squared)
$a\#\% \wedge b\#\%$	
$a\#\% \wedge 12.225$	
$16 \wedge 0.5$	(the square root of 16)
$27 \wedge 0.33333333$	(the cube root of 27)
$3 \wedge -2$	($1/3 * 1/3$)

Modulo arithmetic

MOD gives the remainder when the operand on the left is divided by the operand to the right, e.g.:

$17 \text{ MOD } 7$	(returns 3)
$a\#\% \text{ MOD } b\#\%$	

Parentheses

Parentheses are used to change the order in which the different parts of an expression are calculated. Without parentheses, expressions are evaluated in a certain order which depends on the operators they contain - some operators have precedence over others (see the Table of Precedence). For example, multiplication has precedence over addition; so, in the expression:

$$3 + 4 * 2$$

the 'four times two' part is worked out first and then added to 3, giving 11 as the result. You could alter this by enclosing '3 + 4' in parentheses:

$$(3 + 4) * 2$$

to give a result of 14.

Parentheses must always be paired – each '(' should have a matching ')'.

Concatenation

When used with strings or string variables, the plus sign causes the string specified by the right operand to be appended to the string specified by the left operand, e.g.:

"ABCD" + "defg" evaluates as "ABCDdefg"

Used in this way, the plus sign acts as a string operator.

Relational Operators

Relational operators compare the values of two numbers or two strings, and return a result which is either true or false. They are mainly used with IF THEN and WHILE WEND statements to make a decision about whether the program takes a particular action or not. For example:

```
IF n#% > 30 THEN GOSUB label
```

The relational operators are:

>	greater than	12 > 55 is false, i.e. 0
>=	greater than or equal to	
<	less than	
<=	less than or equal to	27 <= 27 is true, i.e. -1
<>	not equal to	
=	equal to	1 = 5 - 4 is true, i.e. -1
LIKE	pattern matching operator for strings	
CONTAINS	pattern matching operator for text files	

As far as the user is concerned the result of comparison is either true or false. But when DML comes to evaluate a comparison, it actually assigns a numeric value to the result. If the result is true, DML assigns it the value of -1; if false, it assigns the value of 0. This means that if you enter the statement:

```
? 8 > 5
```

or

```
? ("F" = LEFT$("FRIDAY",1))
```

DML will display -1. If the result of a comparison is false, it will display 0.

For string operands, comparison is character by character from the left where 'a' is less than 'b' is less than 'c' and uppercase characters are less than lower case characters ('A' is less than 'a').

Note that numeric data can only be compared with numeric data and string data can only be compared with string data. If you attempt to compare numeric data with string data then the message:

Data types don't match

appears.

LIKE

LIKE is a case insensitive pattern matching relational operator which is used to compare string data. The question mark '?' and asterisk '*' characters can be used as wildcards. The question mark matches single characters and the asterisk '*' matches multiple characters.

For a full explanation of how to use LIKE, see Chapter 10, Volume 1.

CONTAINS

CONTAINS works in the same way as LIKE, but it is used with ASCII files on disk; for example, you could use the expression:

CONTAINS "*Smith"

to search for an ASCII file which contained the name Smith.

Logical or Boolean Operators.

Logical operators are most frequently used to modify the results of relational operators. An expression that contains a relational operator can either be true or false. With a logical operator you can create more complex expressions whose truth or falsity (their 'truth value') depends on the truth or falsity of one or more relational expressions.

AND

This operator allows you to test whether two comparisons are true at the same time. For example:

$12 > 5$ and $6 < 16$

are both true, so:

$12 > 5$ AND $6 < 16$

is also true.

NOT

NOT reverses the truth or falsity of an expression. Thus:

$12 < 5$

is false, so

NOT $12 < 5$

is true.

NOT is often used with the EOF function to set up a program loop for processing all the records in a file, e.g.:

```
WHILE NOT EOF("address")
SELECT NEXT
VIEW
WEND
```

OR

OR gives a true result if one or the other, or both, of its operands is true. For example:

```
12 > 5 OR 12 < 5
8 = 2*3 OR 6 > 5
8 = 2*4 OR 6 < 5
```

all give true results.

```
12 > 5 OR A$ = "OXFORD"
```

gives a true result whatever the value of A\$.

```
8 = 2*3 OR 12 < 5
```

gives a false result.

Logical operators can also be used with numeric expressions to give a result which depends on the binary digits of the operands. In this context, they are sometimes known as bitwise operators. Instead of comparing the truth or falsity of expressions, AND and OR – when used as bitwise operators – compare the binary digits of two numbers, while NOT takes a single number, and changes each binary 1 to 0, and vice versa. Thus AND returns binary 1, if and only if, the corresponding digits in the two operands are both 1, e.g.:

```
85 AND 19
```

gives a value of 17.

```
85 in binary is 01010101
```

```
19 in binary is 00010011
```

The result is 00010001 – which is 17.

Table of Precedence

Complex expressions, containing more than one operator and several simple expressions, are evaluated according to the following table. The higher up the table an operator is, the higher the precedence it is given. This means that if there are two simple expressions, the one whose operator has the highest precedence will be evaluated first.

^	Exponentiation
* /	Multiplication and Division
MOD	Modulo arithmetic
+ -	Addition and Subtraction
< > = <= >=	Relational operators
LIKE CONTAINS	Relational operators
NOT	Logical NOT
AND	Logical AND
OR	Logical OR

Where identical weighted operators are concerned evaluation will be from left to right e.g.:

12 + 3 - 2	is evaluated as	12 + 3 = 15 : 15 - 2 = 13
12 + 3 * 4	is evaluated as	3 * 4 = 12 : 12 + 12 = 24
12 + 3 * 4 - 2	is evaluated as	3 * 4 = 12 : 12 + 12 = 24: 24 - 2 = 22

Parentheses can be used to promote expressions up the table, e.g.:

(12 + 3) * (4 - 2)	evaluates as	15 * 2 = 30
--------------------	--------------	-------------

Constants

The fixed values that you assign to variables or use as part of an expression are known as constants. There are two types: string constants and numeric constants. String constants must be enclosed in quotation marks.

DML also provides a ready-defined constant PI, which has the value of 3.141593265359 and can be used in numeric calculations.

Expressions

Expressions are formed from any combination of:

- Fields
- Variables
- Constants

Functions

Operators

An expression can consist of a single variable name, a field name, or a constant, or it may be more complex, combining these with operators, constants and functions. The following are examples of expressions:

```

396
TODAY
a$
"31 Ambleside Drive"
textfielda
numfielda
Firstname.address
LEFT$(Firstname.address, 1)
3.44 * 22.8 + 450
"Mr " + A$
A$ + B$
a#% <> b#%
(a$ = b$) AND (c#% = 1)
INT(RND(1)*26) + 1
COS(n#%)

```

Line Format and Labels

There are no line numbers in DML; instead, to provide a reference for GOTO and GOSUB commands (or ON GOTO and ON GOSUB commands) you can place a label at the front of a line. Labels can be any combination of alphanumeric characters but cannot contain spaces.

Labels must end in a colon (':') unless they consist only of numeric characters. You can use a number as a label without adding a colon, but this will not have the effect of altering the sequence of program lines. So, if you enter a line with the label '30' after a line with the label '50,' both lines will remain in that order.

A DML line, whether in direct mode or program mode, can contain more than one statement, provided the statements are separated by colons. Labels, however, can only be used in program mode.

Any command which is placed at the beginning of a multi-statement line should be followed by a space. If it is followed by a colon without an intervening space, DML may interpret it as a label. For example, in the line:

```
CLS:STATUS
```

the command CLS has no effect, but:

```
CLS :STATUS
```

works as intended.

You also need to insert a space before an expression – a variable name or a field name, for example – if it is preceded by a DML command word. But otherwise, in program mode, DML will accept lines without any spaces in them (it inserts spaces when it parses a line).

In direct mode, it is generally necessary to insert spaces between the various elements of a command line.

Syntax

The Keyword Reference section gives the exact syntax for each keyword. However, there are several rules that apply generally.

File Names and Field Names

Wherever a command is followed by a file name – or takes a file name as one of its arguments – the file name must be enclosed in quotation marks.

Fieldnames should not be in quotation marks.

For most operations, you should not include the extension name for a file (the part of the name after the period). Superbase automatically looks for files with the appropriate extension; so, with the command:

```
OPEN FILE "Address"
```

it looks for the file **Address.sbf**.

The one exception to this rule is when you are dealing with text files. **OPEN FOR INPUT/OUTPUT**, **OUTPUT TO**, **IMPORT** and **EXPORT** all take data to or from text files, and require that file names are given in full.

If you save a file from Superbase's text editor, it automatically adds an **.SBT** extension to the file name – and when you open a file from the text editor, you do not need to supply the **SBT** extension. But when you make use of a text editor file in some other application – with **IMPORT**, for example – you must remember to include the **SBT** extension.

Upper Case and Lower Case

Certain string functions such as **INSTR** are case sensitive, but otherwise DML ignores the difference between lower and upper case letters (small letters and capital letters). In other words, it does not matter whether you use lower or upper case letters when you type in a keyword or any of the following:

- variables
- file names
- field names

It is a good idea, though, to enter program lines and direct command lines in lower case. Once DML has accepted a line, it converts any reserved words (that is, any

keywords) to upper case. Typing in lower case allows you to check that you have not used a reserved word by mistake.

SELECT Commands

A number of SELECT commands are described in the Keyword Reference Guide, later in this manual. Several of the SELECT commands are closely related, and certain information applies to all of these. That information follows in this chapter.

Record Selection by SELECT Commands

The following SELECT commands relate to the selection of a specific single record in an open database file:

```
SELECT CURRENT
SELECT DUPLICATE
SELECT FIRST
SELECT FORM NEXT/PREVIOUS PAGE
SELECT FORM ROW
SELECT KEY
SELECT LAST
SELECT NEXT
SELECT PREVIOUS
SELECT REMOVE
SELECT WHERE
```

SELECT commands can only be used on an open database file, although this does not have to be the current file.

These commands do not display records on screen. To do this, you need to use VIEW. Similarly, although they can be used with any open file, the SELECT commands do not automatically make an open file the current file.

For example, SELECT LAST selects the last record in a file (on index) even if the file is not current. If the file is current, executing the VIEW command will be enough to display the last record. But with any other open file, you will also need to use the FILE command (as opposed to the FILE parameter) before you can display the record; FILE makes an open file the current file.

When a form (created in the Form Designer) is used to display record data, selecting a record becomes more complicated because the form may contain records from multiple files at the same time. To simplify matters, Superbase provides an optional parameter, FORM, for use with the record SELECT commands. The effect of this parameter is to activate the form's link structure. If it is not used, the SELECT commands only operate on the current file and ignore any links between files.

As an example of how the FORM parameter works, suppose you have created a form which displays data from two files, Clients and Deposits, and also links them together via the Lastname field in each file. Deposits is the master file and becomes the current file when the form is opened from within Superbase. Initially, the file link ensures that the record from the Clients is the one with the same last name as the current Deposits record. But if you now issue a SELECT NEXT command, Superbase will select the next Deposits record (in indexed order) without selecting its corresponding record in the Clients file. SELECT FORM NEXT, on the other hand, takes the link structure into account and will select both the next Deposits record and the Clients record associated with it.

Query Commands

DML's Query commands allow you to create a single multi-line statement which is similar to a Superbase query. These commands are used in report programs created by the Form Designer; but they can also be used for any of the query applications described in Volume 1 – sorting records, creating complex multi-file filters, merging files, and so on.

In Superbase, the Process Query Edit command allows you to define a query with four command lines. For each of the four command lines in the Query Definition requester, DML provides an equivalent query language command. In fact, the command lines take almost exactly the same form in a program as they do in the requester. The only difference is that each line must be preceded by the appropriate query language command.

SELECT is used to define the Fields command line; REPORT defines the Report command line; WHERE corresponds to the Filter line; and ORDER is used for the Order line.

You will find an explanation of these commands under their respective keyword entries. Here, we will describe how they work together to form a query section.

A query section must start with the SELECT command and it should end with END SELECT. Any other query commands are optional; you will include them according to your requirements. Thus, if you wish to use a filter, you will include a WHERE command within the query section. Likewise, if you wish to use reporting functions such as SUM and COUNT, you will need to insert a REPORT command after the SELECT command and before END SELECT.

When REPORT is used in a multi-line SELECT statement it is limited to the syntax provided in the Query Definition requester. If you want to create a report with more extensive statistical or summary detail, you should remove the REPORT line from the multi-line SELECT statement and instead create AFTER GROUP and AFTER REPORT sections as is done by the Form Designer when it generates a report program. (See Chapter 10 Generating Reports.)

You can think of REPORT, WHERE and ORDER as modifying the query output which is specified with the SELECT command. When SELECT is used on its own – to form a single line query section – it outputs data from each record in a specified file (or files) in turn. For example:

```
SELECT Lastname.Address, Country.Address
END SELECT
```

This outputs a line showing the contents of the fields Lastname and Country for each of the records in the Address file. As such, SELECT works in the same way as the ? command except that it acts on all the records in a file.

If you inserted TO PRINTER on a line before END SELECT in the example above, SELECT would output data to the printer. The TO *device* parameter provides an equivalent to the Output options in the Query Definition requester. You can use it to specify an output device other than the screen: the printer, an ASCII file, or a new SBF file.

Converting a Superbase Query to a Select Statement

Any query created with the Query Definition requester can be reproduced under program control. We can illustrate this by converting a query file (an SBQ file) into a program. A typical query might look like this:

```
SB
CLIENTS
DEPOSITS
Deposit Transaction Report
ON "Clients" Firstname.Clients, Lastname.Clients, ON "Deposits"
@24Bank.Deposits, Amount.Deposits, Deposits.Deposits
REPORT SUM Amount.Deposits COUNT GROUP Lastname.Clients
SUM Amount.Deposits
Lastname.Clients = Lastname.Deposits
Lastname.Clients
```

The second and third lines contain the names of the database files which are associated with this query. 'Deposit Transaction Report' is the query title, and the remaining lines represent the four query command lines.

In order to load this query into the Program Editor, we need to change its name. If it was called Deptran.s bq, you would select COPY from the Utilities menu and, after selecting Deptran.s bq as the file to be copied, enter the name Deptran.s bp.

You can now load the file into the Program Editor using the Project Open command. Converting it to a program is just a matter of deleting two lines and inserting keywords in the others; you should also place an END SELECT statement at the end. Once you have made these changes, the program should look like this:


```
OPEN FILE "CLIENTS"  
OPEN FILE "DEPOSITS"  
SELECT ON "Clients" Firstname.Clients, Lastname.Clients, ON  
"Deposits" @24Bank.Deposits, Amount.Deposits, Deposits.Deposits  
REPORT SUM Amount.Deposits COUNT GROUP Lastname.Clients  
SUM Amount.Deposits  
WHERE Lastname.Clients = Lastname.Deposits  
ORDER Lastname.Clients  
END SELECT
```

Note that the REPORT line is the same as in the query file and does not need to be altered.

Running this program would have the same effect as running Deptran from the Query Definition requester.

For more details, refer to the description of the SELECT command in Chapter 17.

Form Creation Using DML

A number of ADD, CREATE and SET commands are described in the Keyword Reference Guide, later in this manual, which together form a group of related commands. Certain information applies to all the commands in the group, and that information follows in this chapter.

The purpose of this group of commands is to allow the creation of forms by DML programs. These commands are normally used together as shown in the example program that appears at the end of this section.

The form creation commands greatly extend the power of the DML programming language. Applications may be customized to allow different forms to be created according to the requirements of the moment, for both input and output. The parameters governing the creation of such an ad hoc form may be stored in a database file, and retrieved when needed to recreate the form.

Reports may be created as forms, incorporating the full range of object attributes available in the Form Designer, and printed one page at a time.

- Forms may not be saved as SBV files; form parameters may be stored as data and retrieved when required.
- Form objects may not be deleted or modified after they have been added to a form.
- Existing SBV files may be used as templates.

Command Summary

ADD FORM	Adds an object to a form
ADD FORM REPLICATE	Defines objects as part of a transaction line
CREATE FORM	Creates page one of a new form
CREATE PAGE	Creates and appends a new page
SET FORM ATTR	Sets object attributes, line and pattern type
SET FORM BF	Turns boldface on or off
SET FORM BG	Sets background (paper) color
SET FORM FG	Sets foreground (pen) and outline colors
SET FORM IT	Turns italic on or off
SET FORM PAGE	Selects page to which objects are added
SET FORM TEXT	Selects font
SET FORM UL	Turns underline on or off
SET FORM USING	Sets units of measurement

Note

The SET FORM commands may be combined in one statement providing that the following order is observed:

```
SET FORM [USING] [PAGE] [FG] [BG] [ATTR] [TEXT] [UL] [BF] [IT]
```

Example Program

The following form creation program begins by opening two data files and setting the units of measurement to millimeters. SET FORM ATTR defines borders for the fields that are to be added. The next section creates a new form "demo" with a link between the data files, and adds two fields from the ADDRESS file. The third section adds six transaction lines, and the program ends by displaying the first master file record.

```
OPEN FILE "adtrans"
OPEN FILE "address"
SET FORM USING 0
SET FORM ATTR 4

CREATE FORM "test",203,254 WHERE
    number.address = number.adtrans
ADD FORM 6,10,10,35,4,number.address
EXAMPLE = ADD FORM 6,50,10,35,4,lastname.address

ADD FORM REPLICATE ON
FOR i% = 1 TO 6
    ADD FORM 6,10,20 + (i% * 10),35,4,serial.adtrans
    ADD FORM 6,50,20 + (i% * 10),35,4,tcount.adtrans
NEXT
ADD FORM REPLICATE OFF
SELECT FORM FIRST
FORM 1
```

16 THE PROGRAM EDITOR AND COMMAND LINE

Superbase's Program Editor provides facilities for the development of programs in DML code. The editor has menus for the management of Superbase SBP program files, Cut and Paste commands, and Search commands.

Most of the Program Editor window's Project menu is reproduced in the Database window as the DML menu, allowing the user to manipulate program files without having the Program Editor window open.

Managing Programs

This section lists the Program Editor's Project menu commands, and gives a brief explanation of their functions.

Command	Opens the Command line requester, allowing the user to enter program instructions as direct commands. See later in this chapter for further information.
Run	Runs the program in memory or, if there is no program in memory, lists programs from which you may select one.
New	Opens the Program Editor window and clears any program that may be in memory.
Open	Lists programs in the current directory. When you select one, Superbase loads it into the Program Editor.
Close	Clears the program in memory – the program that is currently in the Program Editor – and closes the Program Editor window.
Edit	Opens the Program Editor window.
Save	Saves the program in memory to disk.
Save As	Saves the program in memory to disk under a new name.
Remove	Deletes a program file from disk.
Print	Prints the program in memory.
Exit	Leaves the Program Editor window without clearing the program in memory.

The Program Editor Window

When you select Edit or New from the DML menu, Superbase opens a window. This is the Program Editor window where you enter and edit programs.

In most respects, the Program Editor window functions in the same way as the Text Editor window: you use the same controls for closing, resizing and scrolling the window, and you also use many of the same key controls for editing a program.

See Volume 1, Chapter 23 Editing Text for a full description of the facilities of the Text Editor.

- Both the Program Editor and the Text Editor make use of the same window. The window can only be occupied by one of them at a time, but any changes you make to the size or location of the window will apply when you switch to the other editor.

As with the Text Editor, you can switch between the program window and the database window simply by clicking in the one you want to use. Thus, to edit a record after opening the program window, you would click in one of the fields displayed in the database window. If you then wanted to return to the Program Editor, clicking once in the program window would make it active again.

Notice that the Close option on the Project menu not only closes the window but also removes the current program from memory. If you wish to close the window without clearing the program from memory, use the Exit command.

Note

Screen output from a Superbase program is displayed in the database window. If there is a record in view, program data will be output below or to the right of the bottom line of record data. To clear the database window before displaying program data on screen, use the CLS command.

Creating a New Program

Selecting Edit from the DML menu opens the Program Editor window and places the insertion point at the top left-hand corner. When you enter the Program Editor for the first time in a session, the window is empty; you can then start entering program lines straight away. If you have previously loaded a program into memory and want to start afresh on a new program, select New to clear the existing program from memory.

Entering program lines is simply a matter of typing them in at the keyboard. When you have typed in a line, pressing the ENTER key will take you to the start of the next line.

Superbase accepts program lines up to 255 characters long. As you type characters beyond the right-hand edge of the window, the program will be scrolled to the left.

Using the scroll bars enables you to move the window over any part of a line up to 255 characters.

Line Format

An explanation of the correct format for a Superbase program line is given in previous chapter. However, it is worth repeating the rule about inserting spaces in a line: you need to insert a space if you enter an expression – such as a variable name, a field name, or a constant – after a keyword, or if you enter two adjacent keywords. In the following example, the space between the two items on the line must be entered by the user:

```
? "Hello"
```

You should also type a space between a command and a colon if the command is placed at the start of a multi-statement line. If you do not do this, DML will interpret the command as a label.

Otherwise, you can ignore spaces as you type in a program. If a line contains adjacent variables or operators and other elements of an expression, DML inserts spaces between them when it parses the line.

Parsing refers to the process which takes place when you press ENTER or move the cursor away from the current line. DML scans the line and identifies the various elements in it. As well as inserting spaces between recognizable elements, it also converts any keywords that are entered in lower case to upper case. Thus if you typed:

```
if a%=3.5+6then goto label1
```

DML would parse this line as:

```
IF a% = 3.5 + 6 THEN GOTO label1
```

You are recommended to enter program lines in lower case as a way of ensuring that you do not use reserved words as labels or variables. When a line is parsed, you will be able to see at a glance whether these items are correct, because the reserved words will be shown in upper case.

It is unlikely that you will need to create programs with lines over 200 characters long. However, if you do, it is important to bear in mind that Superbase may insert extra spaces in the line. Even though the line you type in is less than 255 characters, it may exceed this limit when it has been parsed.

Editing a Program

The Program Editor uses the same controls for editing a program and moving the cursor around the screen as the Text Editor (see Volume 1, Chapter 23), with the following exceptions:

- There is no provision for margins settings or ruler line.
- There is no Reformat command.
- The text style settings are not available.

Otherwise, the only difference is in the way the ENTER key works. In the Text Editor, ENTER either moves the cursor to the start of the next line or creates a new line, depending on whether overwrite or insert mode has been selected. In the Program Editor, ENTER only creates a new line when you press it at the beginning or end of the existing line.

In this section, we only provide a list of the Program Editor key controls together with a brief description of their functions.

Moving the Insertion Point

With the Program Editor, you can move the insertion point around the screen and edit text at any point on the screen.

You can use the mouse to position the insertion point within the text window. Move the pointer to the point where you want the insertion point to be, and click once. You cannot move the insertion point beyond the last line of text.

There are two ways of bringing text into view if it extends beyond the text window. Moving the insertion point in one direction to the edge of the text window scrolls the window in the same direction. Alternatively, you can scroll the window using the pointer and the scroll bars.

You may prefer not to use the mouse much while you are typing. There is a full set of keyboard equivalents for moving the insertion point.

When you want to:

Press:

Move to the left one character	LEFT
Move to the right one character	RIGHT
Move up one line	UP
Move down one line	DOWN
Move to the start of the line of text	SHIFT+LEFT
Move to the end of the line of text	SHIFT+RIGHT
Move right to the next multiple of 8 characters	TAB
Move left to the next multiple of 8 characters	SHIFT+TAB
Move to the beginning of the text	CTRL+B
Move to the end of the text	CTRL+G

Basic Editing Controls

Most simple editing tasks can be performed with these controls only. Some of these tasks are explained in more detail below.

Split line	CTRL+S
Join lines	CTRL+A
Switch between Insert and Typeover	CTRL+V
Delete to end of line	CTRL+E
Delete to end of word	CTRL+W
Delete current line	CTRL+D
Clear current line	CTRL+X
Undo latest edit	CTRL+U

Cut and Paste

The Program Editor's cut and paste facilities are the same as those in the Text Editor. They allow you to delete, move and copy any section of a program – from a single statement to an entire program – either within the same file or between another program or a text file.

Mark Block	AMIGA+M
Clear Block	AMIGA+K
Cut	AMIGA+X
Copy	AMIGA+C
Paste	AMIGA+V

To mark a block, you may also use the doubleclick method as described in Volume 1. Marking a single letter in a keyword selects the whole keyword as a block.

Using the Command Line

Selecting the Command option on the Program Editor Project menu (or on the DML menu) presents you with the DML command line requester. Any commands you type in the command line box will be executed as soon as you click OK or press ENTER; that is, they will be executed as direct commands.

The command line box accepts single or multi-statement lines (with statements separated by colons) up to 255 characters. There is very little restriction on which keywords can be entered as direct commands. READ (and DATA), GOSUB, and ON GOSUB cannot operate as direct commands, and you cannot use labels in the command line. Apart from these, almost any single or multi-statement line that can be entered in a program can also be entered in the command line.

You will use the command line for variety of purposes. There are some operations – such as appending a text or program file to a file in memory – which can only be performed using commands. You may also want to enter commands as an

alternative to selecting commands from the Superbase menus; but it is often more practical to do this by assigning the commands to a function key.

In addition, the command line is a valuable tool for developing and testing programs. Particularly useful are commands such as:

? MEMORY

and

? LIST

The first command lists the current program's variables and their values; the second can be used to display the program listing in the database window.

Executing a Program from a Label

Clearly, any statement that refers to another statement in the command line will not operate, because of the absence of labels. However, it is possible to enter GOTO statements which refer to labels in the current program. You may want to do this when you are testing a program. The normal way of executing a program – using the RUN command or Run option – clears the program's variables. GOTO allows you to execute a program or part of a program without destroying any variable assignments that may have already been made.

Editing the Command Line

The left and right direction keys move the insertion point in the command line one character at a time. SHIFT+LEFT and SHIFT+RIGHT take the insertion point to the beginning and end of the command line. DEL deletes the character in front of the cursor, BACKSPACE deletes the character before.

If you make a mistake while typing in the command line, you can use these keys to correct it. Command lines remain in the box after they have been executed and function key commands are placed in the command line box after execution, so you may also want to recall the previous command for editing.

Loading a Program

To load a program from disk into the Program Editor, select Open from the Project or DML menu. A file requester appears, listing the names of the program files in the current directory. You select a file in the usual way.

Open loads a program into the Program Editor, but it does not open the Program Editor window. If the window is closed and you wish to edit a program after loading it, select Edit.

To append a program on disk to a program in memory, select Command from the Project or DML menu. Then enter:

`LOAD "programe",APPEND`

where 'programe' is the name of the program on disk which is to be joined to the current program.

Using ASCII Program Source

Most of the programs you run will have been created in the Program Editor. However, it is also possible to load a program which has been written in a word processor, or has been created in another program editor. In this case, there are two requirements:

- The program file must have an SBP extension.
- It must be an ASCII file.

Saving a Program

The Save and Save As commands store the current program on disk. Save uses the current name, Save As allows you to enter a new name.

- The SBP extension is supplied automatically.

Saving ASCII Program Source

Normally, programs are saved on disk in 'tokenized' form; instead of storing each character in a keyword, Superbase represents keywords using single character tokens (keywords are represented in this way internally as well as on disk). To save a program as an ASCII file, enter in the command line:

`SAVE "programe", TEXT`

ASCII program files can be loaded back into the Program Editor and run straight away.

- The SBP extension is supplied automatically.

Running a Program

Select Run from the Project or DML menu to run the program in memory. If there isn't a program in memory, Run displays the program file opening requester. When a program has been selected, it is loaded into memory and then executed.

- Keyboard shortcut: AMIGA+G

The DML command RUN provides the program equivalent of the menu option. It is generally entered as a direct command, but it can also be used within a program. As a program command (rather than a direct command), RUN allows you to perform a task by linking together a series of programs. In this way, you can allocate different stages of a task to different programs. When one stage has been completed, RUN followed by a program file name will load and execute the program which performs the next stage.

However, the drawback to using RUN for this purpose is that loading and executing a new program clears the previous program's variables. If you want to link a number of programs together, it is usually preferable to use the CHAIN command. This provides the same facility as RUN but has the advantage that it does not clear any existing variable assignments.

Note that apart from the Stop and Pause buttons, the browsing controls are disabled while a program is running, but you may re-activate them using the WAIT PANEL command. Stop is only disabled if the BREAK OFF command has been issued; the Pause button is available at all times.

Creating a Start Up Program

One of Superbase's most useful features is the way it lets you specify a 'start up' program, which will be automatically loaded and executed when you run Superbase. There are many ways in which you can take advantage of this feature, especially if you use Superbase on a regular basis for a specific application. All the tasks that you need to perform at the start of a session – such as opening files, loading a function key set, or setting up user menus – can be taken care of by the start up program.

The procedure for creating a start up program is straightforward. There are two requirements:

- The program must be stored in a directory where Superbase will find it: either the Start directory as specified in System Options on the Set menu (see Volume 1, Chapter 32) or the directory which is current when Superbase is loaded; usually this will be in the same directory as the Superbase program file.
- The program must be saved under the file name **Start** (this appears on disk as **Start.sbp**).

When you run Superbase, it looks in the current directory for a program with the name **Start.sbp**. If you have specified a Start directory in System Options, it then looks for the program in that directory.

Examples

```
LOAD KEY "Funkey1"  
OPEN FILE "Address"  
OPEN FILE "Customers"  
SET TABLE  
VIEW
```

This program loads a function file, opens two database files, sets Table View, and then displays the first record in the **Customers** file.

```
LOAD KEY "Funkey1"  
NUMBASE "z999999."  
RUN "Prog2"
```

After loading a function key file, this program sets the numeric format and then loads and executes **Prog2.sbp**.

17 KEYWORD REFERENCE GUIDE

Syntax Conventions

This section describes the conventions that have been used in the following documentation.

The general format used to explain each keyword is as follows:

Purpose – a brief description of the keyword.

Syntax – the format of the keyword.

Comments – explains the format and function of the keyword in more detail.

Examples – program examples.

Notes – explains the program examples and gives any other necessary information.

See Also – lists related keywords.

For the syntax of the keyword, the following terms and symbols are used:

nexpr

This can be any valid numeric expression consisting of numeric fields, date fields, numeric variables, numeric constants and/or numeric results of other keywords.

For example, if a function expects a numeric argument, you could enter any of the following expressions:

156	(an integer)
156.25	(a floating point number)
num%%	(an integer numeric variable)
num#%	(a real numeric variable)
numfield	(the name of a numeric field in a file)
3 + 6	(the result of an expression)
SQR(25)	(the result of a function)

A numeric expression may contain a mixture of numeric types (integer, long and real). Values are 'promoted' if necessary during the evaluation of the expression to avoid overflow, and the final value is truncated if necessary, depending upon its destination.

strexpr

This can be any string or substring from a text field, external field, string variable, text literal or string result of other keywords.

For example, any of the following expressions would be suitable entries for *strexpr*:

"hello"	(a string constant)
strfield	(the name of a string field)
numfield	(the name of a numeric field)
fielda\$	(a string variable)
Address	(a file name)
MID\$("immediately",3,2)	(a calculated result)

var

This can be any variable, string or numeric. If the keyword requires variables of a particular type, *strvar* or *nvar* are used for string or numeric variables.

Various other conventions – such as *fieldname* or just *field* – are used to indicate particular data types. These should be self-explanatory. *sbfile*, for example, means that you should enter the name of a database file; on disk, Superbase identifies database files by an SBF extension, but you should not include the extension name when you use it in a DML command.

Any parameters or arguments contained in parentheses are compulsory (including any commas), while those within square brackets – '[' and ']' – are optional. All other arguments are compulsory. For instance, supposing the syntax for a keyword is given as:

XXX(*strexpr*, *nexpr*[, *nexpr2*])

This means that *strexpr* and *nexpr* are required with the keyword XXX and that they must be separated by a comma. The next argument, *nexpr2*, is optional, but if used must be separated from the second argument by a comma.

The '/' character is used to indicate a number of alternatives. For example:

CLOSE ALL/FILE *sbfile*

The syntax for this statement indicates that CLOSE can be followed by either ALL or FILE, but not both at the same time.

? Commands

? is a general purpose output command. You can use it to display information on screen, or to send information to the printer, or to create a file on disk.

When ? is followed by a list of expressions – text, numbers, variables, fields – it sends the expressions to the current output device. It can also be used with the following keywords:

DIRECTORY
LIST
MEMORY
QUERY
STATUS
TEXT

These keywords have a specific function when used with ?, and have been treated as separate commands. You will find an entry for each of them further on in this section.

Changing the Output Device

When you load Superbase, the ? command takes the screen as the default device and works in the same way as the PRINT command does in other versions of Basic – it displays information on screen. If you want to direct output to another device use one of these commands first:

Parameter	Purpose
PRINT	for the printer
OUTPUT TO	for the disk drive
DISPLAY	to switch back to the screen

? then continues to output to the device selected by one of these commands. CLS and HOME do not reset the current output device.

PRINT and DISPLAY can be followed by a list of expressions which will then be output to the devices associated with these commands. When you use PRINT and DISPLAY simply to change the current output device, they should be followed by a semicolon, as in:

PRINT;

which directs all future output to the printer.

OUTPUT TO needs a filename after it, as in:

OUTPUT TO "texta"

which directs output to a file on disk named 'texta.'

Output Format Parameters

After outputting a list of expressions, the `?` command starts a new line unless you specify otherwise. This means that it takes the cursor to the beginning of the next line (if the current output device is the screen). You can instruct DML to output the next list of expressions on the same line, by placing a semicolon at the end of an expression list. When it is used in this way, the semicolon is known as an output format parameter, and is one of a number of parameters that allow you to specify how output is formatted.

Apart from the semicolon, you place these parameters in front of the expression or list of expressions that you want to format. Thus you can place them at the front of the line immediately after the `?` character, and you can also insert them in the middle of an expression list.

Some output parameters are keywords such as `UL` and `DOWN`, and others are symbols such as the semicolon itself and the `@` sign. If you use keyword parameters, they must be separated from each other by spaces. For example, DML will accept a line like this:

```
? UL IT "One", "TWO" UL OFF @5,10"Three"
```

but it will not accept

```
? ULIT"One", "Two"ULOFF@5,10"Three"
```

The output format parameters for the `?` command are:

Parameter	Purpose
&	Sets length of text string and formats numeric expressions
&[nn]	Left aligns item within a column <i>nn</i> characters wide
& nn	Centers item within a column <i>nn</i> characters wide
&]nn	Right aligns item within a column <i>nn</i> characters wide
@	Sets column and row position
,	Outputs a space
;	Used as a separator and to suppress new line
DOWN	Outputs each item on a new line
ALL	Outputs all the fields in the current record
NEWLINE	Starts a new line
BF [ON]	Sets text to boldface
UL [ON]	Underlines text
IT [ON]	Sets text to italic
BF/UL/IT OFF	Turns selected text attribute off
ATTR OFF	Turns all text attributes off
FG 1/2/3/4	Sets foreground color
BG 1/2/3/4	Sets background color
EJECT [nn]	Outputs data on a new page if there are less than <i>nn</i> lines left at the bottom of the page

Comments

With a text string or text expression, the & parameter sets the maximum number of characters that are displayed. For example:

? &4 "Database"

displays 'Data' and truncates the rest of the word.

When it is used with a numeric expression, & allows you to specify the format for numeric output. As a numeric format parameter, & must be followed by a decimal number. The first part of the number (the integer part which occurs before the decimal point) specifies the number of digits that will be displayed before the decimal point; the second part (after the decimal point) specifies the number of decimal places for the numeric output. For example:

? & 5.3 (2/3)

displays:

0.667

The above example also shows you what to do if the decimal number following the & is itself followed by a numeric constant. A space is not a sufficient separator; it is necessary to put the numeric expression in brackets.

Three modifiers are available for the & formatting parameter to specify left, right, or centered output of a field or variable. For example, to force a date to be left aligned in a column 20 characters wide, use a command of the form:

? &[20 datefield.FILENAME

Proportional fonts are not supported with this feature.

The modifiers may also be set from within the Report Generation part of the Form Designer. To specify the type of alignment required, select the item or items required, and click the justification tool in the toolbox to the left, vertical (centered), or right pointing position.

The @ sign, followed by a pair of numeric expressions separated by a comma, specifies the column and row position for your output. For example, if the screen is the output device selected:

? @12,7 "Smith"

displays the name 'Smith' on the screen at column 12, row 7. If no row number is specified the current row is used. As an alternative to using @, see LOCATE. See also HOME.

DML treats a comma as a space character. For example:

? "Hello",,"goodbye"

outputs

Hello goodbye

Placing a semicolon after the last expression in the output list prevents DML from outputting a new line character. For example:

```
? "Hello": ? " there"
```

outputs 'Hello' and ' there' on separate lines.

```
? "Hello";: ? " there"
```

outputs them on the same line as

```
Hello there
```

If used, DOWN must be placed at the front of a line, before the line's expressions and after the ? character. It ensures that each expression in the list is output on a new line. For example:

```
? DOWN "one";"two";"three"
```

formats the output on screen as:

```
one  
two  
three
```

When DOWN is used to format field data, it displays the field names in inverse video at the left of the data – in same way as Record View displays a record.

ALL outputs the fields in the current record; for example, if the screen is the current output device:

```
? DOWN ALL
```

would have the same effect as clicking on the current record button when Record View is selected.

NEWLINE instructs DML to start a new line. Thus

```
? "one", "two" NEWLINE "three"
```

outputs the expressions as

```
one two  
three
```

NEWLINE can also be used on its own – see NEWLINE, Keyword Reference.

BF, UL and IT set the text style for output.

BF	sets boldface
UL	sets underline
IT	sets italic

Each of these can be set by ON (which is optional) and cleared by OFF.

ATTR OFF clears all the text style attributes. For example:

? UL "Underline on"; IT "italics on"; UL OFF IT OFF "Underline and italics off"

FG and BG set the foreground and background colors in which text is shown in the work area. The values 1, 2, 3, and 4 correspond to the background of the window, the color of the text in the window, the background of the menu bar, and the background of the title bar.

? FG 4 BG 3 "hello"

prints 'hello' in dark blue text on a cyan background, assuming the standard default colors for the title and menu bar.

EJECT can be used to ensure that a group of data items – the fields in a record, for example – all appear on the same page. For example:

? DOWN "One";"Two";"Three";"Four" EJECT 4

displays a new page if there is not enough room at the bottom for the four items in the expression list.

?

Purpose

Sends information to a device.

Syntax

? [*@nexpr,nexpr*] [*expr,expr;...*]

Comments

Used to send one or more expressions (an expression list) to the current output device. ? can also include any of the output format parameters.

When you output numeric, date, and time constants or the results of functions for these data types, they take the format which has been set with the Number or Date/Time format options on the Set menu (or with the commands NUMBASE and DATEBASE). Field data is output in the format set in the file definition. However, you can force Superbase to output field data in the current format by enclosing the field name in brackets.

Another way of formatting numeric data is to use the STR\$ function's format parameter. The same options are also available for date and time constants using the DATE\$ and TIME\$ functions.

Examples

- 1 ? "Hello"
- 2 ? "Hello "; a\$
- 3 ? UL "Hello "; a\$; NEWLINE b\$
- 4 ? "The square root of "; n#%; " is "; SQR(n#%)
- 5 ? "One""two""three"

Notes

Depending on the numeric format that has been set, example 4 would output the numeric expressions, n#% and SQR(n#%), with leading spaces. To remove these, you would have to use the STR\$ and LTRIM\$ functions.

You will notice that there are no semicolons separating the expressions in example 5. It is not always necessary to place a semicolon between two expressions. Provided DML has some other way of telling where one expression ends and the other one starts, you can omit the semicolon.

See Also

OPEN, CLOSE, HOME, LOCATE, SELECT

? DIRECTORY

Purpose

Lists the current directory to an output device.

Syntax

[?] DIRECTORY

Comments

When the screen has been selected as the current output device, the directory is listed in two columns in alphabetic order. If the output device is the printer or a disk file, the directory is listed in single column.

The "?" command is optional – DIRECTORY on its own has the same effect.

See Also

DIRECTORY

? LIST

Purpose

Lists a program in memory to an output device.

Syntax

? LIST

Comments

Do not confuse this command with LIST, which displays a text file on screen. ? LIST has several applications. You can use it to examine a program listing by displaying it on screen or printing it out; and, when the current output device is the disk drive, it allows you to save a program as a text file. In this last application, it has the same effect as:

```
SAVE "program name",TEXT
```

See Also

LOAD, SAVE TEXT

? MEMORY

Purpose

Sends information about any defined variables to an output device.

Syntax

[?] MEMORY

Comments

This outputs the variables currently stored in memory. For example:

```
i#% = 123.46' Real
j%% = 12' Integer
k&% = 123456000' Long
a1$ = "aaaaaaaaaaaaa bbbbbbbb"
etc.
```

The variables are output in the order in which they were assigned. If you specify OUTPUT TO *file*, the values of the variables can be recovered for a subsequent program by the SET *file* command. The '?' command is optional – MEMORY on its own has the same effect.

See Also

ERASE, CLEAR, LET, DIM, SET

? QUERY

Purpose

Runs the current Query in memory and sends the results to an output device.

Syntax

? QUERY [TO *device*]

Comments

There are four options for query output. As well as the screen, you can send the output to the printer, to disk (as an ASCII file) or to file (as an SBF file). The *TO device* parameter specifies one of these three devices. If this parameter is not included output is to the screen. The device options are:

TO PRINTER

Outputs to the printer.

TO FILE filename

Creates an SBF file on disk under the file name specified.

TO filename

Outputs to the ASCII file specified by 'filename.'

Examples

- 1 LOAD QUERY "Stockrep"
? QUERY TO PRINTER
- 2 LOAD QUERY "Addreport"
? QUERY TO "Addrep.asc"
- 3 ? QUERY TO FILE "new-adds"
- 4 ? QUERY

Notes

In the first example, output from the Stockrep query is directed to the printer; in the second example, output from the Addreport query is used to create an ASCII file on disk under the name Addrep.asc. Example 3 uses the current query to create a new database file under the name 'new- adds.' The last example outputs the query in memory to the screen.

See Also

LOAD, SELECT

? STATUS

Purpose

Sends information about the System Status.

Syntax

[?] STATUS [FILE *sbfname*]

Comments

? STATUS without the FILE option gives the same output as the Utilities menu command Status System . If no Superbase files (SBF files) are open, it displays the following information:

- Memory free
- Diskspace Free
- Current Directory Name
- Superbase Files open
- Superbase Files available.

If any database files are open, the header information for these files is also shown.

? STATUS FILE *sbfname*

gives the same output as the menu command Status File, and produces a status report for the file. At the top of the report, it shows file statistics such as the number of records and the file size. Below this it shows the file definition details: for each field, it lists the field name, its attributes, format, Form View location, and any associated calculation or validation formulas.

If the file is not open, the error message 'File not open' appears.

The '?' command is optional with STATUS – using STATUS on its own has the same effect.

See Also

RECCOUNT, FILE, INDEX

? TEXT

Purpose

Sends a text file in memory to an output device.

Syntax

? TEXT [MERGE]

Comments

This command outputs the current text file in the Text Editor. If the text file is a form letter and its associated database file has been opened, adding the MERGE command will merge the data in the current record with the text file. See also the Keyword Reference entry for MERGE.

Examples

- 1 OPEN FILE "Address"
LOAD TEXT "Maillet"
? TEXT MERGE
- 2 PRINT;
? TEXT
DISPLAY;

Notes

If the current output device is the screen, the first example will insert the data from the first record (in the Address file) into the Maillet text file, and display the result in the database window. Example 2 sends the current text file to the printer.

See Also

LOAD

ABS

Purpose

Turns negative numbers into positive numbers but leaves positive numbers unchanged.

Syntax

ABS (*nexpr*)

Comments

ABS returns the 'absolute' value of a number. In effect, it simply strips the sign off a negative number, making it positive. One of its common uses is for calculating the difference between two numbers when you do not know which is larger. In example 2, which works out the number of days between two dates, ABS allows you to enter the dates without first knowing which is the later date.

Examples

- 1 numfieldc = ABS(numfielda)
- 2 numfieldc = ABS(datefielda - datefieldb)
- 3 IF numfieldc > ABS(numfielda * numfieldb) THEN ...
- 4 x&% = ABS(y&%)
- 5 x#% = ABS(y#% * numfielda * (datefielda - datefieldb))
- 6 x%% = ABS(VAL(RIGHT\$(textfielda, 5)))
- 7 ? ABS(x#%)

See Also

SGN

ADD

Purpose

Adds a new field to a file definition.

Syntax

ADD [FILE*sbfname*] *field-definition-string* [*formula-string*] [*formula-string*]

Comments

This command is used in conjunction with CREATE and MAKE to create a new Superbase database file. It cannot be used to add a field to an existing file.

To use ADD correctly, you need to understand how a file definition is set up. The best way to do this is to use the STATUS FILE option in Superbase's Utilities menu. This will show you the file definition for the current file. Another way of viewing the same information would be to list an SBD file on screen using the Utilities menu's LIST option.

With one exception, the parameters used with ADD to define a field are the same as those used in SBD file definitions. The exception is '>' and '>>'. These are used in SBD files to indicate that there are one or two formula lines to follow. You do not need to include these characters in a field definition string.

The field definition string contains the parameters described below, in the order given and separated by semicolons.

fieldname

fieldtype [other fieldtype parameters]

format [case and multiple response parameter for text fields]

row

col

Each of these parameters must take the same form as it does in an SBD file.

fieldname must conform to the rules for fieldnames.

fieldtype must be one of the following:

TXT	text
EXT	external
NMI	numeric (integer)
NML	numeric (long)
NUM	numeric (real)
DAT	date
TIM	timefield
LOG	logical
VTX	Virtual text field
VNI	Virtual numeric (integer) field
VNL	Virtual numeric (long) field

VNU	Virtual numeric (real) field
VDA	Virtual date field
VEX	Virtual external field
VTI	Virtual time field
VLO	Virtual log field

The other fieldtype parameters are optional and, if used, are separated from *fieldtype* and each other by at least one space. You can specify up to three of these parameters, selecting one from each of the following groups:

A)	
CLC	Calculated
CLV	Calculated and validated
CON	Constant
COV	Constant and validated
VAL	Validated
B)	
RDO	Read only
RDQ	Read only required
REQ	Required
C)	
IXU	Indexed - Unique
IXD	Indexed - Normal

format specifies the length of a field and the numeric format for numeric fields. *format* must be appropriate for the field type. For example, 20 for TXT, z99999.00 for NUM, ddmmyy for DAT.

The case parameter can be U, L or C, depending on whether the text data is to shown in upper case, lower case or with the first letter capitalized. The multiple response parameter is M followed by the number of elements in the multiple response field (maximum 9). These parameter are optional and can only be used with text fields. They must be separated from the field format by at least one space. For example, to define a text field with three multiple response fields, each up to 15 characters long, you would enter these parameters:

15 M3;

If you also wanted the field data to be in upper case, you would enter:

15 UM3;

row defines the row number of the position in Page View (and has a maximum value of 255).

col defines the column number of the position in Page View (and has a maximum value of 255).

row and *col* are optional, but you are strongly recommended to use them. If they are not used, all the fields will be positioned at the top left of the screen on top of one another when the record is displayed in Page View.

The field types in group A – CLC,CLV,CON,COV,VAL – must be followed by a formula or formulas. CLV and COV require two formulas; the formula for constants or calculations should come first, followed by the validation formula. Formulas should be specified by means of valid text string expressions, and must be separated by commas from the first part of the field definition string and from each other.

formula-string must follow the rules (given in Volume 1) for using formulas for calculations, constants and validations.

Since formulas must be enclosed in quotation marks, DML will be confused if a formula itself contains quotation marks, as in:

```
fielda LIKE "[a-c]"
```

The simplest solution here is to use the tilde character '~' in place of quotation marks. When you execute the ADD command, DML will interpret the tilde as a quotation mark.

An alternative solution is to use the CHR\$ function to insert quotation marks – the ASCII code for quotation marks is 34. Thus the line to set up a validated field called fielda of length 12 at row 5, column 20 would be:

```
ADD "fielda;TXT VAL;12;5;20","fielda LIKE " + CHR$(34)
+ "[a-c]" + CHR$(34)
```

In the unlikely event that your formula actually requires a tilde character, you can insert it in a string variable using CHR\$(126).

Note that the maximum length for a program line is 255 characters. Since a formula can also be up to 255 characters long, you may be unable to enter the entire field definition on a line. The solution is to assign formulas to string variables beforehand. For example:

```
a$ = "Price.Orders * Quantity.Orders"
ADD "Subtotal;NUM CLC;$9999999.00;5;20",a$
```

Examples

- 1 ADD "aatext;TXT REQ IXU;20 M3;1;12"
- 2 ADD "aanumb;NUM;z999999.00;1;40"
- 3 ADD "bbtext;TXT CLC RDO;10;2;12", "UCASE\$(LEFT\$(aatext,10))"
- 4 ADD "fielda;TXT VAL;12;5;20", "fielda LIKE ~[a-c]*~"

Notes

Creating a file under program control is a three stage process involving CREATE, ADD, and MAKE. See the entry for CREATE.

See Also

CREATE, MAKE, MODIFY, SAVE FILE

ADD FORM

Purpose

Adds an object to the current form page.

Syntax

ADD FORM *nexpr,parameter list*

Comments

The ADD FORM command adds an object to the page specified with the SET FORM PAGE command.

Nexpr defines the object type:

Type	Object
1	Areas
2	Boxes
3	Lines
4	Images
5	Text
6-10	Fields, Calculations, Commands, Checkboxes, Radio buttons

The *parameter list* varies according to the object type.

In the syntax variants given below, the following notation is used:

Argument	Meaning
<i>x</i>	The x co-ordinate of the object in current units
<i>y</i>	The y co-ordinate of the object in current units
<i>width</i>	The width of the object in current units
<i>height</i>	The height of the object in current units
<i>strexpr</i>	A text string
<i>field</i>	The field associated with the object
<i>var</i>	The variable associated with the object
<i>number</i>	The data entry order number of the field.

Note If the *number* parameter is zero then the next highest available entry order number will be given to the object.

Syntax for AREAS: object type 1

ADD FORM 1,*x,y,width,height*

Syntax for BOXES: object type 2

ADD FORM 2,*x,y,width,height*

Syntax for LINES: object type 3

ADD FORM 3,*x,y,width,height*

Width should be 1 for a vertical line. *Height* should be 1 for a horizontal line.

Syntax for IMAGES: object type 4

ADD FORM 4,*x,y,width,height,image name*

Image name is a string expression containing the path and filename of the image required.

Syntax for TEXT: object type 5

ADD FORM 5,*x,y,stexpr*

Stexpr is a string expression containing the text string to be added to the form.

Syntax for FIELDS: object type 6

ADD FORM 6,*x,y,width,height,field,number*, [*stexpr*]

Stexpr is a string expression containing a validation formula for the field.

Syntax for CALCULATIONS: object type 7

ADD FORM 7,*x,y,width,height,var, number* [,*stexpr*]

Stexpr is an optional string expression containing a calculation to be performed.

Syntax for COMMANDS: object type 8

ADD FORM 8,*x,y,width,height,var, number,stexpr*

Stexpr is a string expression containing a command to be executed.

Syntax for CHECKBOXES: object type 9

ADD FORM 9,*x,y,width,height,field/var,number,stexpr1,stexpr2*

Field/var defines the field or variable to which the checkbox is related. *Stexpr1* contains a string whose content is stored in the field or variable when the checkbox

is selected. *Strexpr2* contains a string whose content is stored in the field or variable when the checkbox is *not* selected.

Syntax for RADIO BUTTONS: object type 10

ADD FORM 10,*x,y,width,height,field/var,number,strexpr*

Field/var defines the field or variable to which the radio button is related. *Strexpr* contains a string whose content is stored in the field or variable when the checkbox is selected. @MINOR HEADING = See Also

ADD FORM REPLICATE, CREATE FORM, CREATE PAGE, SET FORM, and the section Form Creation Using DML in Chapter 15.

ADD FORM REPLICATE

Purpose

Causes field objects added subsequently to be part of a transaction line.

Syntax

ADD FORM REPLICATE ON/OFF

Comments

ADD FORM REPLICATE ON should be executed before you define transaction lines – the ‘many’ part of a one-to-many form structure.

ADD FORM REPLICATE is normally followed by a loop which executes an ADD FORM command for each object in the transaction line. The number of times the loop itself is executed determines the number of transaction lines created.

ADD FORM REPLICATE OFF should be executed when all the transaction lines have been created.

Example

```
ADD FORM REPLICATE ON
FOR a%% = 1 to 10
  ADD FORM 6,10,10+(a%% * 10),40,8,fld1
  ADD FORM 6,60,10+(a%% * 10),40,8,fld2
NEXT
ADD FORM REPLICATE OFF
```

The example adds two fields per line, at x co-ordinates 10 and 60, with a width of 40 and a height of 8. The ‘10+(a%% * 10)’ expression uses the loop variable a%% to control the y (vertical) co-ordinate for the fields. Since the FOR...NEXT loop executes 10 times, 10 transactions will be added.

See Also

ADD FORM, CREATE FORM, CREATE PAGE, SET FORM, and the section Form Creation Using DML in Chapter 15.

AFTER GROUP

Purpose

Set after group activity with reports.

Syntax

AFTER GROUP *fieldname*

Comments

When you create a report using the Form Designer's reporting facility, Superbase generates a Report program for you. The AFTER GROUP statement is used within a Report program to mark the start of an AFTER GROUP section. This consists of a series of '?' statements which specify the information that is output every time a group changes. Each statement corresponds to a line in the AFTER GROUP box in the Form Designer.

fieldname identifies the group and must be the name of a field which has previously been defined as a group with the GROUP statement. An AFTER GROUP section must end with an END GROUP statement.

Examples

```
1  AFTER GROUP Lastname.Clients
   ? "Client "; GROUP; " has "; COUNT; " deposits"
   ? "Total deposits are"; @25 SUM amount
   ? "Maximum is"; @25 MAX amount
   ? "Minimum is"; @25 MIN amount
   END GROUP
```

Notes

In the second line of this example, the keyword GROUP is used in order to retrieve the name of the last group. At this point, the group has already changed, so Lastname.Clients could not be used instead of GROUP because it would output the name of the current group.

See Also

END GROUP, REPORT, SUM

AFTER REPORT

Purpose

Marks the start of an AFTER REPORT section in a Report program.

Syntax

AFTER REPORT

Comments

This command must be placed at the head of an AFTER REPORT section in a report program. Like the AFTER GROUP section, this contains a series of '?' statements which output statistical information about specified fields. The difference is that the information relates to field data for the whole report and is only output once, at the end of the report.

Examples

```
1  AFTER REPORT
   ? @1,3; UL "Statistics for the Report"; UL OFF: ?
   ? "Total amount is"; @30 SUM amount
   ? "Number of deposits"; @30 COUNT
   ? "Average amount"; @30 MEAN amount
   END REPORT
```

See Also

END REPORT, REPORT, SUM

AFTER SELECT

Purpose

Allows processing following output of SELECT data during a report.

Syntax

AFTER SELECT

Comments

Reports may require that additional processing is performed as each record is selected during the execution of a SELECT statement. The BEFORE SELECT and AFTER SELECT statements allow you to trigger such processing either before or after the output of the fields defined in a SELECT statement. Both require an END SELECT statement to terminate the section.

Examples

```
1  AFTER SELECT
    IF Amount.CLIENTS = 0 THEN
        Flag.CLIENTS = "Y"
        STORE FILE "CLIENTS"
    END IF
END SELECT

2  AFTER SELECT
    EJECT
END SELECT
```

Notes

The first example assigns a value to a field and updates the record if the Amount field is zero, so that a subsequent operation can select all the updated records easily even if the Amount field is no longer zero.

The second example shows how to print one record per page.

See Also

BEFORE SELECT

ASC

Purpose

Returns the ASCII value of a single character.

Syntax

ASC (*strexpr*)

Comments

ASC gives a numeric value – the ASCII code – for the text character in *strexpr*. When *strexpr* is longer than one character ASC returns the ASCII code of the leftmost character of *strexpr*.

The complementary function of ASC is CHR\$.

Examples

```
1  numfieldc = ASC(textfielda)

2  numfieldc = ASC(RIGHT$(textfielda, 1))

3  IF ASC(textfielda) >= ASC("A") AND ASC(textfielda) <= ASC("Z") THEN .
..
```

- 4 x%% = ASC("A")
- 5 x%% = ASC(x\$)
- 6 x%% = ASC(UCASE\$(MID\$(extfield, 3, 1)))
- 7 ? ASC(x\$)

Notes

Example 3 provides one way of validating a text field to ensure that the first character is uppercase (but see FCASE\$).

See Also

CHR\$, FN ANSI, FN IBM

ASK

Purpose

Inputs a character string from the keyboard.

Syntax

ASK [*pos*] [*string*] [*&length*] ;*var*/*field*

Comments

This command allows the user to enter information from the keyboard into the computer while a program is running. It expects the user to type in one or more characters, and, as soon as the ENTER key is pressed, reads them into a variable or a field.

pos gives the position of the prompt text and input string. It uses the @ output format parameter to specify the column position, followed by the row position. If *pos* is not given then the prompt text (if any) and input string start from the current cursor position. *string* can be used to include a prompt message. *&length* limits the length of the input and shows the maximum length on screen using an 'end-of-field' marker.

The input can be assigned to a variable or a field (in a currently open file). If the field is not in the current file, it can be specified with the filename extension.

var or *field* can be either string or numeric variables or fields, and they must match the type of input expected: a string variable will accept whatever you type in, but a numeric variable expects numeric data. DML assigns a value of zero to numeric variables when you input non-numeric data such as letters. The same restriction applies to numeric fields.

Examples

- 1 ASK; x\$
- 2 ASK @2,12 "Enter a text string "; x\$
- 3 ASK @2,14 "Enter a 3 character word " &3; x\$
- 4 ASK "Enter data – Customer Code " &8; Cuscode.custfile
- 5 ASK "Enter a number "; n%

See Also

CLS, GET, HOME, INPUT, LOCATE, WAIT

ATN

Purpose

Calculates an angle whose tangent is known. The result is given in radians.

Syntax

ATN (*nexpr*)

Comments

This function works out the angle from the angle's tangent given in *nexpr*. For example: TAN (0.7854) is 1.0, so ATN (1.00) returns 0.7854 rads (which is 45 degrees). To convert radians to degrees multiply by 180/PI.

ATN is the complementary function of TAN.

Examples

- 1 numfieldc = ATN(numfielda)
- 2 x#% = ATN(y#%)
- 3 x#% = ATN(VAL(x\$))
- 4 ? ATN(x#%)

See Also

COS, SIN, TAN

BEFORE GROUP

Purpose

Set before group activity with reports.

Syntax

BEFORE GROUP *fieldname*

Comments

BEFORE GROUP is used in a Report Program to mark the start of a BEFORE GROUP section. *fieldname* identifies the group, and must be the name of a field which has previously been defined as a group – in the Form Designer, you define a group in the SELECT box; Superbase then translates this into a GROUP statement in a Report program.

The statements in the BEFORE GROUP section are executed every time the group changes. Typically, they would be used to display header information for the group data which follows. A BEFORE GROUP section must end with an END GROUP statement.

Examples

```
1 BEFORE GROUP Lastname.Clients
  ? "Deposits for client", Lastname.Clients
  ? "Firstname Lastname Bank Amount Date"
  ? "-----"
END GROUP
```

See Also

AFTER GROUP, END GROUP

BEFORE SELECT

Purpose

Allows processing prior to output of SELECT data during a report.

Syntax

BEFORE SELECT

Comments

Reports may require that additional processing is performed as each record is selected during the execution of a multi-line SELECT statement. The BEFORE SELECT and AFTER SELECT statements allow you to trigger such processing

either before or after the output of the fields defined in a SELECT statement. Both require an END SELECT statement to terminate the section.

Examples

```
1  BEFORE SELECT
    IF Amount.CLIENTS = 0 THEN flag$="Balance is zero"
    ELSE flag$=""
  END SELECT
  SELECT Name.CLIENTS,Amount.CLIENTS,flag$
```

Notes

After the record has been read into memory the program checks the value of the Amount field and assigns a message to the variable flag\$ before outputting any data. If flag\$ is one of the items in the SELECT line then it will print a message if the client's balance is zero.

See Also

AFTER SELECT

BELL

Purpose

To attract the user's attention.

Syntax

BELL

Comments

Use this command to signal that a particular process has finished and to attract the user's attention. BELL causes the screen to flash.

BLANK

Purpose

Creates a blank record in memory.

Syntax

BLANK [FORM] / [DUPLICATE] [FILE *sbfname*]

Comments

When you want to create a new record under program control, issuing a BLANK statement is the first step in the process. It sets up an empty record ready for data entry. There are several ways in which you can then enter data into the record's fields.

Using BLANK with ENTER allows the user to type in the data for a new record from the keyboard. Together these two commands have the same effect as the RECORD NEW option in the RECORD menu. Alternatively, you can enter data from within a program, by assigning it directly to the fields in the new record. Typically, this would be the method you used to create new records by reading data in from another file on disk.

The FORM parameter is used with a multi-file form; BLANK FORM creates blank records for each file represented in the form. When a form is open but the FORM parameter is not included, BLANK only creates a blank record for the current file.

DUPLICATE is the program equivalent of Duplicate on the Record menu. It creates a copy of the current record and allows the user to create a new record by editing the copy. This parameter can only be used with a file, not a form.

With the FILE parameter, you can create a blank record in a file which is open but not current.

Examples

- 1 BLANK
- 2 BLANK FILE "abc"
- 3 BLANK
ENTER
STORE
- 4 BLANK
Firstname.address = "John"
Lastname.address = "Smith"
STORE

5 BLANK DUPLICATE
ENTER
STORE

The first example creates a new record in the current file. Example 2 creates a new record in another open file. Example 3 creates a new record in the current file, and allows the user to enter data in the record; then it saves the record on disk. Example 4 creates a new record, and enters data into the fields Firstname and Lastname; then saves the record on disk. Example 5 allows the user to create a new record by editing the current record.

See Also

CLEAR, ENTER, STORE

BREAK

Purpose

Allows the user to interrupt or halt a program from the keyboard.

Syntax

BREAK [ENTER] ON / OFF

Comments

BREAK ON and BREAK OFF enable and disable the Stop button and CTRL+C. BREAK ON is the default condition and allows the user to stop a program by pressing CTRL+C or clicking on the Stop button. CTRL+C generates error 11. If you want to include error handling routines in your programs, you can use the ON ERROR statement to check for this error number.

BREAK OFF prevents the user from stopping a program; it also prevents the space bar from being used to pause during program execution. When a program is finished, BREAK is turned on again.

The ESCAPE key method of terminating data entry may be disabled by including the ENTER option. CTRL+C can still be detected after BREAK ENTER OFF has been executed, unless BREAK OFF has been executed separately. BREAK ENTER ON re-enables ESCAPE data entry termination.

Unless the BREAK OFF command has been executed, CTRL+C can be detected during data entry. CTRL+C generates error number 11, which may be trapped following the ON ERROR GOTO command.

CALL

Purpose

Executes an AmigaDOS command.

Syntax

CALL strexpr [WAIT] [CLOSE]

Comments

strexpr is a string expression containing a command line to be executed by AmigaDOS. The expression should duplicate a command line that would normally be typed at the CLI prompt. Superbase stays in memory so there must be enough memory for the specified command to function correctly.

You may use CALL in two ways to execute AmigaDOS commands. The first way is to enter:

```
CALL "newcli"
```

This opens a CLI window on the Workbench screen, allowing you to type in and execute any of the AmigaDOS commands.

The second way of using CALL allows you to specify the required AmigaDOS command directly. For example:

```
CALL "dir dh0:"
```

For CLI operations, the CALL command can take two extra parameters, WAIT and CLOSE.

WAIT causes Superbase to wait for you to press RETURN in the Superbase CLI window before returning.

CLOSE closes the Superbase CLI after the CALL is complete. If you CLOSE the window, you will need to WAIT first, to give time to see the CLI output.

You may want to use CALL to run a program that requires input from the keyboard. In this case, you should specify '<*' after the program file name, as in:

```
CALL "fcopy <*"
```

The effect of this is to redirect input from 'stdin', allowing you to enter data in the CLI window.

The CALL command may also take additional parameters for use with the ARexx macro language (see Chapter 19).

See Also

ARexx, SET POSITION FOR

CHAIN

Purpose

Executes another program from within a program but does not clear the first program's variables.

Syntax

CHAIN *filename*

Comments

CHAIN allows you to carry out a task by linking together a number of programs. When a program is running, this command loads another program from disk, and transfers control to it. The second program displaces the first in memory but any variables that have already been set retain their values.

Examples

```
1  CHAIN "Nextprog"
```

See Also

CLEAR, LOAD, RUN

CHR\$

Purpose

Generates the character associated with an ASCII code.

Syntax

CHR\$ (*nexpr*)

Comments

CHR\$ works in the opposite way to ASC. Whereas ASC takes a character and returns its ASCII code, CHR\$ generates the character from its associated code. It is useful for handling characters which are not available from the keyboard, such as certain characters used to control a printer.

nexpr must have a positive value in the range 0 - 255 (although not all these values will give printable characters). A value outside this range will give the error 'Invalid numeric parameter.' If *nexpr* is not an integer, the integer part of the number is used, i.e., 65.999 is treated as 65.

Examples

```
1  textfieldc = CHR$(numfielda)
2  textfieldc = CHR$(INT(numfielda / 256))
3  x$ = CHR$(65)
4  x$ = CHR$(ASC(y$) + 32)
5  FOR n%% = 32 to 127
    ? CHR$(n%%);
  NEXT
```

Notes

Example 3 stores the letter A in x\$. Example 4 demonstrates one way of turning an uppercase character into lowercase – but see LCASE\$ and FCASE\$. Example 5 displays the character set on the screen. Note that it does not display characters for codes below 32; the reason for this is that most of these characters are non-printing characters, and some of them have unexpected effects when you attempt to display or print them.

See Also

ASC

CLEAR**Purpose**

Clears all user variables.

Syntax

CLEAR

Comments

This command clears all variable assignments in memory. Using ? MEMORY immediately after CLEAR would give no output.

The CLEAR statement must be placed last on a program line.

See Also

DIM, ERASE, RUN

CLOSE COMMS

Purpose

Closes the communications channel.

Syntax

CLOSE COMMS

Comments

This command should be executed at the end of any operation involving data transfer.

See Also

COMMS ?, COMMS FILE, COMMS FILE GET, COMMS GET, COMMS INPUT, OPEN COMMS, WAIT COMMS

CLOSE FIELDS

Purpose

Closes the field list on the current file or another open file.

Syntax

CLOSE FIELDS [FILE *sbfname*]

Comments

CLOSE FIELDS removes any restrictions on which fields are shown. If you VIEW a record after issuing this command, all its fields will be displayed on screen.

On its own, CLOSE FIELDS closes the field list for the current file. When FILE *sbfname* is added, it closes the list for that file.

Examples

1 CLOSE FIELDS

Close field list on current file.

2 CLOSE FIELDS FILE "abc"

Close field list for file 'abc.'

See Also

OPEN FIELDS

CLOSE FILE

Purpose

Closes all files or a specified file. Use this command to release memory for other use, or to avoid confusion if you want to open a file of the same name in a different directory.

Syntax

CLOSE [ALL] / [FILE *sbfname*]

Comments

CLOSE FILE works in the same way as the equivalent option in the Project menu. When followed by the ALL parameter, it closes all the open files at once.

Examples

1 CLOSE

Closes the current file.

2 CLOSE FILE "aaa"

Closes the database file 'aaa.'

3 CLOSE ALL

Closes all open database files.

See Also

CLOSE FORM, OPEN FILE, OPEN FORM

CLOSE FORM

Purpose

Closes the current Form. Use this command to release memory for other use, or to avoid confusion if you want to open a form of the same name in a different directory.

Syntax

CLOSE FORM

Comments

This command works in the same way as the Form Close option on the Project Menu: it clears the current Form from memory and displays the current file using one of the view modes.

See Also

OPEN FORM

CLOSE INPUT/OUTPUT

Purpose

Closes an input or output channel to a text file.

Syntax

CLOSE INPUT/OUTPUT

Comments

Many database files (and their associated index files) may be open simultaneously. In addition, one input channel and one output channel can be open for access to text (non-database) files. You are advised to use the CLOSE command when you have completed an I/O task involving a text file, otherwise you will be unable to open a new channel.

CLOSE also ensures that all the data in the disk buffer is written to disk. If you have a large disk buffer and a small amount of data, you will notice that some output commands – ? LIST, for example – appear to have no effect. What happens is that DML places the data in the buffer before writing it to disk. It remains there until the buffer fills up – or until you execute a CLOSE OUTPUT command. Associated commands are OPEN and INPUT.

Examples

```
1  OPEN FILE "aaa" FOR OUTPUT
   FOR i%% = 1 to 10: ? i%%, i%% ^ 2: NEXT i%%
   CLOSE OUTPUT
   OPEN FILE "aaa" FOR INPUT
   getnext: INPUT LINE a$: ? a$
   IF NOT EOF(****) THEN GOTO getnext
   CLOSE INPUT
```

See Also

OPEN FOR, OUTPUT TO

CLS

Purpose

Clears the screen.

Syntax

CLS

Comments

CLS makes the output part of the screen – the output window – blank and takes the cursor to the top left-hand corner.

Examples

```
1 CLS : ? "Now at top of cleared screen"
```

See Also

COL, HOME, LOCATE, ROW

COL

Purpose

Returns the cursor's position across the screen.

Syntax

COL (0)

Comments

Use this command to find out the column position of the screen cursor. For the row position, see ROW.

Examples

```
1 x%% = COL(0)
```

```
2 ? COL(0)
```

Notes

This function expects to be provided with a numeric value (the parameter in brackets), even though there is no information for you to supply. You should always specify a value of zero, as shown.

In practice, Example 2 would be pointless because the statement changes the cursor position in the course of printing it.

See Also

HOME, LOCATE, ROW

COMMS ?

Purpose

Outputs characters through the comms channel.

Syntax

COMMS ? *stexpr*

Comments

This outputs the characters in the string expression (either a string variable or a text string enclosed in quotation marks) through the comms channel. For example:

COMMS ? "Hello"

sends the word 'Hello' to the computer at the receiving end.

See Also

CLOSE COMMS, COMMS FILE, COMMS FILE GET, COMMS GET, COMMS INPUT, OPEN COMMS, WAIT COMMS

COMMS FILE

Purpose

Transmit a file using the comms channel.

Syntax

COMMS FILE [?] *stexpr*

Comments

This command is the program equivalent of transmitting a file using the Comms menu option. The string expression should specify the file or files that are to be transmitted. For example, to transmit the Address database files, you could enter:

```
a$ = "Address"  
COMMS FILE ? a$ + ".*"
```

COMMS FILE makes use of the options that are set in the comms requester or with the OPEN COMMS command. For example, if an initialization string has been entered, it sends this to the modem; if an autodial number has been entered and DCD is not high, it dials the number; and if AUTO is on, it sends the file header.

See Also

CLOSE COMMS, COMMS ?, COMMS FILE GET, COMMS GET, COMMS INPUT, OPEN COMMS, WAIT COMMS

COMMS FILE GET

Purpose

Receives a file using the comms channel.

Syntax

COMMS FILE GET *strexpr*

Comments

Use this command to receive a file under program control. The string expression (*strexpr*) should specify the name under which the file is to be saved. However, if the Auto option is on, it will be saved using the name under which it was transmitted; so you do not need to supply the file name and can just give the empty string. For example:

```
OPEN COMMS 0,9600,0,1,8,0,1  
COMMS FILE GET ""  
CLOSE COMMS
```

As well as Auto, COMMS FILE GET makes use of the other options that are set in the comms requester or with the OPEN COMMS command. For example, if an initialization sequence has been entered in the requester, it sends this to the modem before doing anything else.

See Also

CLOSE COMMS, COMMS ?, COMMS FILE, COMMS GET, COMMS INPUT, OPEN COMMS, WAIT COMMS

COMMS GET**Purpose**

Reads a character from the comms buffer.

Syntax

COMMS GET [TEXT] *stringvar*

Comments

This works in a similar way to GET when it is used as a keyboard input command. Instead of reading a character from the keyboard, COMMS GET reads a character from the comms buffer and stores it in the string variable specified. If the buffer is empty it returns a null character. As with GET, it doesn't wait for a character to be transmitted.

If you want COMMS GET to wait for transmission, you need to place it in a loop, as in:

```
lab1: COMMS GET a$: IF a$ = "" THEN GOTO lab1
```

The TEXT parameter limits the range of characters that are returned to those with ASCII codes less than 128. Characters whose codes are greater than 127 are converted to this range (i.e., Superbase strips off the highest bit); for example, ASCII character 180 would be returned as character 52.

See Also

CLOSE COMMS, COMMS ?, COMMS FILE, COMMS FILE GET, COMMS INPUT, OPEN COMMS, WAIT COMMS

COMMS INPUT**Purpose**

Reads characters from the comms buffer.

Syntax

COMMS INPUT [TEXT] *stringvar*

Comments

This command reads up to 255 characters at a time from the comms buffer into the string variable specified. If it finds a line feed character (ASCII 10) it only takes the characters up to and including the line feed. Otherwise, within the limit of 255, it returns as many characters as there are in the buffer.

The optional TEXT parameters operates in the same way as it does with COMMS GET. It maps characters over 127 onto their ASCII equivalents in the range 0 to 127.

See Also

CLOSE COMMS, COMMS ?, COMMS FILE, COMMS FILE GET, COMMS GET, OPEN COMMS, WAIT COMMS

COPY

Purpose

Makes a copy of a file on disk.

Syntax

COPY from.filename [,]/[TO] to.filename

Comments

Use this command either to copy a file to the same disk under a different name, or to copy it to another disk. In the latter case, you can give the file the same name or a new name.

Examples

1 COPY "aaa","bbb"

Copies 'aaa' to 'bbb.'

2 COPY "sys:aaa" TO "dh0:bbb"

Copies 'aaa' from drive sys: to drive dh0: under the name 'bbb'.

See Also

CREATE NEW, DELETE, EXPORT, RENAME, REORGANIZE

COS

Purpose

Returns the cosine of an angle measured in radians.

Syntax

`COS ( nexpr )`

Comments

To convert an angle in degrees to radians, multiply by $\text{PI}/180$. Associated functions are ATN, SIN and TAN.

Examples

- 1 `numfieldc = COS(numfielda)`
- 2 `x#% = COS(ndegs#% * PI / 180)`
- 3 `x#% = COS(VAL(x$))`

See Also

ATN, SIN, TAN

COUNT

Purpose

Returns the number of repeated fields in a report or transaction form. Also returns the number of elements in an array.

Syntax

`COUNT ( fieldname / array )`

Comments

As a report function, COUNT can only be used with repeated instances of a field or a variable such as occur in reports and transactions or in an array. Unlike the other report functions, COUNT works with string, date and time fields as well as numeric fields.

When a calculation variable is replicated as part of a transaction line, it is converted into an array variable. In most cases COUNT will be used with these types of variables – the arrays that occur in a transaction form rather than normal

program arrays. If you supply a program array with more than one dimension as an argument, COUNT operates on the first dimension.

When it is used to count the fields in a block of transactions, COUNT only operates on the lines that contain data. With arrays, it gives the total number of elements irrespective of whether they have been assigned a value.

Examples

- 1 x&% = COUNT trans%
- 2 IF COUNT Lastname.Address > 20 THEN . . .

Notes

Parentheses are optional with COUNT and other report functions.

See Also

DIM, GROUP, MAX, MEAN, MIN, REPORT, SD, SUM, VAR

CREATE

Purpose

Creates a new database file in memory, or splits an existing database file into two files.

Syntax

CREATE *sbfname* [:*passwords*]

CREATE NEW FILE *sbfname* FROM FILE *sbfname*

CREATE FILE *sbfname* AND *sbfname* FROM *sbfname*
[WHERE *conditions*] [:*passwords*]

Comments

CREATE is only the first step in the process of building a new file. To define the file and store it on disk you also need to use ADD and MAKE. The whole process involves the following steps:

1. CREATE *sbfname*
2. ADD *field*

At this point the file is held in memory and you can check it with ? STATUS *sbfname*.

3. MAKE *sbfname*

Writes the file to disk.

Examples

The CREATE NEW FILE option allows you to derive an empty copy of a database structure from an existing one. The CREATE FILE *sbfname* AND *sbfname* FROM *sbfname* option is designed for splitting a database into two separate files. The source file remains intact.

```
1  CREATE "Address"
   ADD "Recno;NML CON IXU;999999.;0;0","SER(~Address~)"
   ADD "Title;TXT;10;1;34"
   ADD "Firstname;TXT;15;3;6"
   ADD "Lastname;TXT IXD;20;3;34"
   ADD "Street;TXT;30;6;6"
   ADD "City;TXT IXD;15;7;6"
   ADD "Code;TXT;12;7;31"
   ADD "Country;TXT IXD;15;9;6"
   MAKE "Address"
```

Notes

This example shows how to set up a simple address file under program control. The first ADD statement defines a numeric field which is automatically assigned a record number by means of the SER function. Note the use of the tilde character to enter the filename Address within quotation marks. Following this, the ADD statements define seven text fields which will hold the name and address.

Once you have defined the file in this way, you could then enter its data under program control, using the commands BLANK and STORE.

See Also

COPY, EXPORT, REORGANIZE, SELECT TO FILE

CREATE FORM

Purpose

Creates a new form structure without objects.

Syntax

CREATE FORM *strexpr* [*width*[,*height*],*hierarchy*]] [WHERE *conditions*]

Comments

CREATE FORM closes the current form if one is open and automatically creates the first page of a new empty form. The *width* and *height* parameters should be given in the units defined with SET FORM USING; if they are not given, the form page is created at the current printer page size.

Hierarchy consists of six parameters specifying the hierarchy of the objects on the first page:

Type	Object
1	Areas
2	Boxes
3	Lines
4	Images
5	Text
6	Fields, Calculations, Commands, Checkboxes, Radio buttons

The default hierarchy is 1, 2, 3, 4, 5, 6 meaning that objects of type 1 (areas) are drawn first followed by objects of type 2 (boxes), etc. By changing the sequence you can change the way in which objects are drawn. For example:

CREATE FORM "test",203,254,1,2,4,3,5,6

This form would be drawn with lines appearing on top of images.

The WHERE clause specifies the file links for the form and should match the layout of the file links that would be set using the form designer. The syntax of the link takes the form:

WHERE field1.FILE_A = field2.FILE_B

If there is more than one link, each such expression must be separated from the next with the AND keyword. *No other keywords are allowed* – this is not a normal filter expression.

Note CREATE FORM does not reset the default values for the SET FORM group of commands. Therefore, if you are creating several forms one after another, you should execute the appropriate SET FORM commands before each ADD FORM command.

See Also

ADD FORM, ADD FORM REPLICATE, CREATE PAGE, SET FORM, and the section Form Creation Using DML in Chapter 15.

CREATE INDEX

Purpose

Creates a new index file.

Syntax

CREATE INDEX ON *fieldname.sbfname* [UNIQUE]

Comments

This command is the program equivalent of the Index New option on the Project menu. *fieldname.sbfname* specifies the field in a particular file that is to be indexed. Use the UNIQUE parameter if you require a unique index.

Examples

1 CREATE INDEX ON aaa

2 CREATE INDEX ON aaa.bbb UNIQUE

'bbb' does not need to be the current file, but it must be open.

See Also

INDEX, REMOVE INDEX

CREATE PAGE

Purpose

Creates a new form page without objects.

Syntax

CREATE PAGE *strexpr* [*width*[,*height*],*hierarchy*]]

Comments

CREATE PAGE creates a new empty page and appends it to any existing pages. The *width* and *height* parameters should be given in the units defined with SET FORM USING; if they are not given, the form page is created at the current printer page size.

The hierarchy parameters function as for CREATE FORM.

See Also

ADD FORM, ADD FORM REPLICATE, CREATE FORM, SET FORM, and the section Form Creation Using DML in Chapter 15.

DATA

Purpose

Holds the data (numeric and string constants) that is accessed by a READ statement.

Syntax

DATA *constant* [,*constant*] [.....]

Comments

The values (constants) following a DATA statement must be separated by commas and text constants must be in quotation marks.

To insert quotation marks in a string constant, either use single quotes (to avoid confusion with the double quotes enclosing the string) or use CHR\$(34).

Date values which are to be read into date fields must be in the correct format (e.g., "dd mm yy" or "mm dd yy") and should be enclosed in quotation marks.

If you intend to use the RESTORE statement with reference to a specific line, that line should begin with a label.

Examples

- 1 DATA "abcde", 1.04, 2.46, "uvwxyz"
- 2 data1: DATA 12.2, 12.4, 12.97, 13.4, 9.2, -1
- 3 DATA "Superbase 'DML' commands"
- 4 DATA "Superbase " + CHR\$(34) + "DML" + CHR\$(34) + " commands"

See Also

READ, RESTORE

DATE\$

Purpose

Returns a text string from a julian date number.

Syntax

DATE\$ (*nexpr* [,*format-string*])

Comments

This function expresses a date number as a text string showing the day, month and year. The *format-string* option is used to specify the date format for the text string. It must be a valid Superbase format as shown in the entry for DATEBASE. If the format is not given, the text string takes the date format as set with the Date Format option on the Set menu, or as specified with the DATEBASE command.

The complementary function to DATE\$ is DAYS. Associated date functions are DAY DAY\$ DAYS MONTH MONTH\$ YEAR.

Examples

- 1 textfieldc = DATE\$(datefielda, "dd mm yy")
- 2 textfieldc = DATE\$(datefielda + 90)
- 3 textfieldc = DATE\$(TODAY)
- 4 x\$ = DATE\$(y&%)

Notes

Example 2 gives the date 90 days after the date in datefielda. Example 3 provides a calculation to insert the system date into a textfield.

Unless you are sure that the date supplied in *nexpr* will always fall within this century, you should set the date format to allow four figures for the year; otherwise you will not be able to distinguish between 1901 and 2001.

With dates before AD 1000, the four figure year number has leading zeros.

Early calendars were not particularly accurate and day counts between dates cannot be relied on before AD 1400. (Superbase always gives the same answer; history is less consistent.)

See Also

DAY, DAY\$, DAYS, DATEBASE

DATEBASE

Purpose

Sets the DATE format and TIME format.

Syntax

DATEBASE *string*

Comments

DATEBASE allows you to specify the format with which Superbase displays the date and time. Normally, this format only applies when you use the date and time functions, DATE\$ and TIME\$, or when you use the system variables, TODAY and NOW. It does not affect the format of date and time fields, as set in the file definition. This means that if you define a date field with the format "dd-mm-yy", the command

? datefield

will display it in this format, irrespective of what the DATEBASE format is. However, you can force Superbase to display a date field in the current date format by enclosing the field name in parentheses, as in:

? (datefield)

string must have a valid date or time format. Valid formats are also shown as options in the Date/Time requester selected from the Set menu.

Single digit days and months are given a leading zero or a leading space as appropriate for the format in use (see the following examples).

Examples

These examples show how a date of September 5th 1990 and a time of 8.345 seconds past 2.35 pm are displayed.

- 1 DATEBASE "dd mmmm yyyy"
5 September 1990
- 2 DATEBASE "dd mmm yy"
5 Sep 90
- 3 DATEBASE "dd/mm/yy"
5/09/90
- 4 DATEBASE "mm-dd-yy"
5-09-90
- 5 DATEBASE "dd.mm.yy"
5.09.90

- 6 DATEBASE "dd mm yy"
5 09 90
- 7 DATEBASE "ddmmyy"
050990
- 8 DATEBASE "hh:mm"
14:35
- 9 DATEBASE "hh:mm:ss"
14:35:08
- 10 DATEBASE "hh:mm:ss.s"
14:35:08.345
- 11 DATEBASE "hh:mm am"
2:35 pm

Notes

Example 8 shows how you can use the DATEBASE command to specify a date format without separators. Such a format is not available from the date/time format requester.

See Also

DATE\$, TIME\$

DAY

Purpose

Returns the day of the month as a numeric value from a date field or a date string.

Syntax

DAY (*expr*)

Comments

The number which DAY returns takes the numeric format as set with Number format menu option, or as set with the command NUMBASE. Associated date functions are DATES DAY\$ DAYS MONTH MONTH\$ YEAR.

Examples

- 1 numfieldc = DAY(datefielda)

- 2 numfieldc = DAY(datefielda + 90)
- 3 numfieldc = DAY(TODAY)
- 4 x%% = DAY(datefielda + VAL(textfielda))
- 5 x%% = DAY(DAYS("11 Jan 85"))
- 6 ? DAY (datefielda + 30)

Notes

Example 3 provides a calculation to insert the day number of the system date into a numeric field.

See Also

DATE\$, DAY\$, DAYS

DAY\$

Purpose

Returns the day of the week as a text string from a julian date number.

Syntax

DAY\$ (*nexpr*)

Comments

Associated date functions are DATE\$ DAY DAYS MONTH MONTH\$ YEAR.

Examples

- 1 textfieldc = DAY\$(datefielda)
- 2 textfieldc = DAY\$(datefielda + 90)
- 3 textfieldc = DAY\$(TODAY)
- 4 x\$ = DAY\$(DAYS("11 January 1985"))
- 5 x\$ = DAY\$(y&%)
- 6 ? DAY\$(datefielda + 30)

Notes

Example 3 provides a calculation to insert the weekday of the system date into a textfield. Example 4 provides a weekday for the date shown in quotation marks.

See Also

DATE\$, DAY, DAYS

DAYS

Purpose

Returns the date as a julian date number from either a text string or a numeric value.

Syntax

DAYS(*strexpr*) or DAYS(*nexpr*)

Comments

This function returns a number which is the julian day number of the date in *strexpr* (31 December AD 0 has a julian date value of zero). It takes the 1752 Gregorian reform of the calendar into account.

strexpr must be in a valid date format.

A text expression which is not in one of the valid date formats will produce a message 'Invalid date format.' Associated date functions are: DATE\$ DAY DAY\$ MONTH MONTH\$ YEAR.

If a numeric expression is given as the parameter to this function, the value returned is simply the result of the expression.

Examples

- 1 numfieldc = DAYS(textfielda)
- 2 numfieldc = DAYS(datefielda + 90)
- 3 numfieldc = DAYS("11 jan 85")
- 4 x&% = DAYS(a\$) + 90

Notes

In example 2, the use of DAYS is redundant.

See Also

DATE\$, DAY, DAY\$

DEBUG

Purpose

Allows single step execution of a DML program.

Syntax

DEBUG ON [,*speed*]/OFF

Comments

This command will be most useful to the developer working with DML programs. It allows you to see which statement in a program is actually being executed at each moment. The statement being executed is shown in the Superbase CLI window.

DEBUG also slows a DML program to a speed at which an observer can follow the execution of individual statements. The *speed* parameter uses units of hundredths of a second.

Examples

```
1  DEBUG ON,100
```

Slows program execution to one statement per second.

DELETE

Purpose

Deletes a file stored on disk.

Syntax

DELETE *filename*

Comments

This command deletes the specified file from disk. To delete a file on a drive or directory other than the current one, you need to place the path name in front of the file name. If you want to delete a database file, use the REMOVE FILE command; unlike DELETE which only deletes a single file at a time, REMOVE FILE deletes all the individual files (SBD, SBF, index files) that make up a database file.

Examples

```
1  DELETE "aaa"
```

```
2 DELETE "dh0:aaa"
```

See Also

COPY, RENAME, REMOVE FILE, REMOVE FROM

DIM

Purpose

Defines array variables

Syntax

DIM *variablename*(*nexpr*[,*nexpr*] [,*nexpr*])

Comments

Arrays can have up to three dimensions. *nexpr* specifies the number of elements in each dimension of an array. The maximum number of elements is limited only by the amount of available memory.

A single DIM statement can be used to define more than one array. If DIM is used in this way, each array definition should be separated by commas.

Note that the first element in an array dimension has the subscript 0.

Examples

```
1 DIM x%%(20)
```

Defines a one dimensional numeric array with 21 elements.

```
2 DIM b$(10)
```

```
3 DIM x#%(3,10)
```

```
4 DIM b$(2,12)
```

```
5 DIM a$(20), n&%(2,3,10), c$(2,10)
```

See Also

CLEAR, ERASE, LET

DIRECTORY

Purpose

Changes the current directory, returns the path of the current directory, displays the contents of the current directory or re-reads the current directory.

Syntax

DIRECTORY "*path*"

Comments

DIRECTORY can be used to switch to a directory on another drive or to a different directory in the same drive (see Example 1).

When used as a system variable, DIRECTORY returns the current path (see Example 2). The command ? DIRECTORY lists the contents of the current directory (see Example 3).

The command DIRECTORY "" forces Superbase to re-read the current directory, so ensuring that any file creations or deletions are included in a directory listing (see Example 4).

Examples

- 1 DIRECTORY "dh0:mydir/testdir"
- 2 y\$ = DIRECTORY: ? y\$
- 3 ? DIRECTORY
- 4 DIRECTORY "": ? DIRECTORY

See Also

? DIRECTORY

DISKSPACE

Purpose

Shows the amount of space remaining on the specified disk.

Syntax

DISKSPACE (*strexpr*)

Comments

This function gives the number of unused bytes on a disk. *strexpr* may be either a drive name or a volume name, and must end with a colon. Note that the free blocks given by the CLI are 488-byte blocks.

Examples

- 1 x\$ = "dh0:" : x% = DISKSPACE(x\$)
- 2 ? "Remaining disk space is: "; DISKSPACE ("dh0:"); " bytes"

See Also

DIRECTORY

DISPLAY

Purpose

Sends information to the screen.

Syntax

DISPLAY [*expressionlist*]

Comments

DISPLAY, followed by a semicolon and nothing else, selects the screen as the current output device. The ? command can then be used to send information to the screen.

Since the screen is the default output device, normally you will only use the DISPLAY command when you want to switch back to the screen after selecting another device such as the printer.

Examples

- 1 PRINT;
? MEMORY
DISPLAY;
- 2 PRINT "This line will print on the printer"
DISPLAY "This line will appear on screen"
PRINT "Another line for the printer"
DISPLAY;

See Also

?, PRINT

EDIT

Purpose

Allows the user to edit a program, a text file, a Query, or an Update.

Syntax

EDIT [TEXT]/ [QUERY]/ [UPDATE]

Comments

Depending on which option has been selected, EDIT displays a window or a requester.

EDIT QUERY displays the query requester.

EDIT UPDATE displays the update filter requester.

EDIT TEXT opens the text editor window.

If none of the three options has been selected, EDIT opens the program window.

After EDIT has opened a window or requester, you can edit the information it presents and insert new information, just as you would if you had selected one of the edit options from the Superbase menus – Query from the Process menu, for example. The difference is that when you exit from the window or requester, control returns to the program.

To exit from a window, close the window by clicking on the close gadget in the top left-hand corner. You exit from a requester by clicking on OK or Cancel. In Superbase itself, clicking on OK takes the user out of the requester, and then runs the query or update; clicking on OK after EDIT returns control to the program without executing a query or an update. (To run a query, use ? QUERY, and use UPDATE to carry out an update.)

If the user clicks the Cancel button on an EDIT QUERY or EDIT UPDATE requester, a CTRL+C error is generated (error 11). This should be trapped and handled with ON ERROR, ERRNO, ERR\$, and RESUME group of commands.

See Also

?, LOAD

EJECT

Purpose

Ejects the current page on the printer or feeds in a new page when the number of lines remaining is less than the number specified.

Syntax

EJECT [GROUP] [*nexpr*]

Comments

EJECT on its own has the same effect as pressing the Form Feed button on the printer. It sends a Form Feed character to the printer, which then moves the current page on and feeds in the next. If *nexpr* is used, the page is ejected when the number of lines at the bottom of the page is less than *nexpr*.

The optional parameter GROUP may be used to suppress the EJECT for the last group in a report. This avoids the unnecessary printing of a blank page after the last group.

Examples

1 EJECT

2 EJECT 3

Feeds in a new page if there are less than three lines left at the bottom of the current page.

See Also

REPORT

END

Purpose

Terminates program execution.

Syntax

END

Comments

This command brings a program to a halt and returns control to the Superbase Menu system. END is optional and if you do not include it, DML will return control to Superbase when it reaches the last statement in a program.

Examples

```
1  IF EOF("aaa") THEN END
```

See Also

QUIT

END FOOTING

Purpose

Marks the end of a FOOTING section in a Report program.

Syntax

END FOOTING

Comments

END FOOTING must be placed at the end of the statements that define the footing information in a report – the information that is to appear at the foot of every page.

See Also

FOOTING

END GROUP

Purpose

Marks the end of a BEFORE GROUP or AFTER GROUP section.

Syntax

END GROUP

Comments

A BEFORE GROUP or AFTER GROUP section in a Report program must end with an END GROUP statement. If it does not, Superbase will be unable to tell which statements belong to which section.

See Also

AFTER GROUP, BEFORE GROUP

END HEADING

Purpose

Marks the end of a HEADING section in a Report program.

Syntax

END HEADING

Comments

A HEADING section defines the headings for a report. It must start with the keyword HEADING, followed by one or more “?” statements which specify the heading information. END HEADING must be placed at the end of the section.

See Also

FOOTING, HEADING, REPORT

END REPORT

Purpose

Marks the end of an AFTER REPORT or BEFORE REPORT section.

Syntax

END REPORT

Comments

END REPORT must be placed at the end of the set of statements that constitute an AFTER REPORT or BEFORE REPORT section in a Report program. Superbase automatically generates END REPORT statements at the end of these sections when you create a report using the Form Designer's reporting facility.

See Also

BEFORE REPORT, REPORT

END SELECT

Purpose

Indicates the end of a query section or a SELECT CASE statement.

Syntax

END SELECT

Comments

END SELECT should be placed at the end of a multi-line SELECT statement (also referred to as a query section) in order to indicate to Superbase where query section starts and where the rest of the program starts. With query statements, you can use blank line as an alternative to using the END SELECT statement.

Examples

```
1  SELECT Firstname, Lastname, Street, City
   WHERE Lastname Like "[a-c]*" AND CITY = "London"
   ORDER Lastname
   END SELECT
```

See Also

AFTER SELECT, BEFORE SELECT, SELECT, SELECT CASE

ENTER

Purpose

Allows the user to enter data in the current file or to edit the data in a record.

Syntax

ENTER [*field1* [ROW *nexpr1*]] / [*nexpr2*] [TO *field2* [ROW *nexpr3*]]
/ [,*nexpr4*]

Comments

ENTER works on the current file. Used on its own, it is equivalent to the Record menu option Edit. It displays the current record and allows you to edit it field by field, starting with the first field that is not Read Only. In effect, ENTER temporarily hands over control from DML to Superbase itself. When you press ENTER after the last field in a record or when you move the insertion point down to the bottom of the record, control is transferred back to DML.

The parameters for this command can only be used with a form. When a form is open and *field1* or *nexpr2* is specified without any further parameters, Superbase restricts the editing to just one field. You can supply either a field name or a field number where the number corresponds to the field's position in the field order; i.e. the entry order which was set when the form was created.

After executing an ENTER command followed by *field* or *nexpr2*, Superbase makes the insertion point active in the field specified; control is transferred back to DML when the user presses ENTER.

With TO or *nexpr4*, you can specify that a range of fields is available for editing. TO must be followed by a field name and restricts editing to the fields in the range *field1* to *field2*. *nexpr4* gives the number of fields that can be edited – from *field1* or *nexpr2* onwards.

As an example, suppose you wanted to restrict data entry from four fields from Firstname through to City. Assuming Firstname is number 2 in the form's entry order and City is number 5, you could use any of the following commands:

```
ENTER Firstname TO City
ENTER 2 TO City
ENTER 2,4
ENTER 2 TO City
ENTER Firstname,4
```

Superbase would make the cursor active in the Firstname field and allow you to enter data or edit Firstname and the next three fields.

You may want to specify a starting point without restricting yourself to a given number of fields. To do this, enter 0 as the second numeric parameter; e.g.:

```
ENTER 2,0
```

or

```
ENTER Firstname,0
```

Now you will be able to edit any field in the form starting from field 2. Control will be transferred to DML when you press ENTER in the last field. This option is the only one that allows you to click in any field.

The ROW parameter is used for entering data in transactions. *nexpr1* specifies the number of the row at which you wish to start entering data, *nexpr3* specifies the last row.

Note that ENTER can also be used to edit form calculations. But in this case you should supply the calculation's order number, not the calculation name. Thus suppose you wished to enter a new value in a calculation named Calc1% which had the order number 7; the correct command to do this would be:

```
ENTER 7
```


Because calculations on a form are treated as variables by DML, the command:

ENTER Calc1%

would only work correctly if Calc1% happened to contain the value of 7.

When used in conjunction with BLANK, ENTER is equivalent to the Record menu option New, and allows you to enter data into a new record. ENTER does not save a record; to save a new or edited record use STORE.

Examples

- 1 OPEN FILE "aaa"
ASK "Record to edit " ; x\$
SELECT KEY x\$
ENTER
STORE
- 2 OPEN FILE "aaa"
BLANK
ENTER
STORE
- 3 ENTER Lastname
- 4 ENTER 4,3
- 5 ENTER Cuscode TO Quantity
- 6 ENTER Cuscode ROW 3 TO Quantity ROW 5

Example 1 shows how to enter an existing record in order to edit it. Example 2 creates a new record. Example 3 restricts editing to the Lastname field. In example 4, fields (or calculations) 4 to 6 in the form's field order can be edited. Example 5 restricts editing to the fields from Cuscode to Quantity. The last example assumes there are transactions on the form and restricts editing to the fields in rows 3, 4 and 5.

See Also

BLANK, STORE

EOF

Purpose

Detects the end of a database file when reading through it under program control.

Syntax

EOF (*strexpr*)

Comments

strexpr should contain the name of a currently open file. If DML reaches the end of a file, EOF is set to -1 (true); otherwise it is set to 0 (false). EOF is only used under program control.

Supplying the empty string as an argument – EOF("") – allows you to refer to the current file without giving the file name.

EOF can also be used to detect the end of a text file. For this purpose it takes the string "*" as an argument, and not the file name; that is, EOF("*") is set to 'true' when the end of a text file is found.

EOF is only set and unset by the SELECT record selection commands excluding SELECT KEY (which only sets the FOUND function). In other words it is possible to search through a file with a filter, and set EOF to 'true' by not finding a record, and then to use key-lookup (SELECT KEY) to look for a record outside the filter. If you do this, FOUND will reflect the success of the key-lookup operation, while EOF will show that you are at the end of the file.

Examples

```
1  ? EOF ( x$ )
   OPEN FILE "aaa": SELECT FIRST:
   IF EOF("aaa") THEN GOTO fempty
   loop1: SELECT NEXT: IF EOF("aaa") THEN GOTO fend
   .....
   GOTO loop1
   fempty: ? "FILE 'aaa' has no records":END
   fend: ? "Process completed": END
```

See Also

SELECT

ERASE

Purpose

Clears a variable assignment from memory.

Syntax

ERASE *varlist*

Comments

This command clears a variable or a list of variables from memory. If several variables are to be cleared, *varlist* should contain a list of variable names separated by commas.

If you specify the name of an array, ERASE clears all the elements in the array (it also clears any other variable with the same name as the array).

Variables in use on a form cannot be erased or cleared. To free them, close the form.

Examples

```
1  ERASE x$
2  ERASE a#%, b&%, c%%, d%, a$, b$
3  a$ = "Fred"
   a#% = 2.35
   MEMORY
   ERASE a$
   MEMORY
```

Notes

The most obvious use for this command is to tidy up the variables in memory before CHAINing another program or before OUTPUT TO "aaa" MEMORY. You could then use SET "aaa" to pass some values to variables in the new program.

See Also

CLEAR, DIM, LET

ERR\$

Purpose

Returns the text message associated with an error number.

Syntax

ERR\$ (*nexpr*)

Comments

This function returns a text string containing the error message associated with the error number given in *nexpr*.

Use this function in conjunction with the control flow commands ON ERROR and RESUME, and the system variable ERRNO.

Examples

```
1  ? ERR$(ERRNO)

2  CLS: NUMBASE "z9999"
   FOR i%% = 1 to 180
   ? "Error number ";i%%;" ";ERR$(i%%)
   NEXT i%%

3  x$ = ERR$(14)
```

Example 2 displays a list of error numbers together with their associated error messages. See Volume 1, Appendix A.

See Also

ERRNO, ON ERROR

ERRNO

Purpose

Returns the number of the last error that occurred.

Syntax

ERRNO

Comments

ERRNO is used in conjunction with the error handling commands ON ERROR and RESUME and with the function ERR\$ which gives the error message associated with an error number.

Examples

- 1 x%% = ERRNO
- 2 ? ERR\$(ERRNO)

Notes

Example 2 displays the error message associated with the last error which occurred. To display a list of all the error numbers and their associated messages, run example 2 in the entry for ERR\$. A list of error messages with their numbers appears at the end of Volume 1, Appendix A.

See Also

ERR\$, ON ERROR

EXECUTE

Purpose

Executes a text string as though it were a command.

Syntax

EXECUTE *string*

Comments

Any set of commands, statements, and functions that can be carried out as a single program line can be placed in a string and then executed. If EXECUTE is entered on a multi-statement line, it must be placed last on the line.

Examples

- 1 EXECUTE "SELECT CURRENT:VIEW:WAIT FOR 5"
y\$ = "REQUEST ~OK/Cancel Requester~,~~,1"
EXECUTE y\$

Notes

Example 1 suggests a way of using this function to set up a user-defined requester which can be called from different points in a program. Note the use of the tilde character to embed quotation marks in a string.

See Also

SET

EXISTS

Purpose

Checks whether a file exists on disk or whether an index value exists in an index.

Syntax

EXISTS (*strexpr* [,*indexfield*])

Comments

You may use EXISTS for two purposes: to check whether a file exists on disk, and to check whether an indexed field contains a specified value. If the file or index value exists, the function returns a value of -1 (True); if it does not exist, it returns 0 (False).

In practice, the file check will generally be used in a conditional statement such as:

IF EXISTS ("tempfile") THEN....

The index value check works on any index of any file, without reading a record.

For example:

EXISTS ("jones",lastname.ADDRESS)

looks up 'jones' in the Lastname index of the Address file, returning True (-1) if 'jones' exists in it, False (0) if it does not. The function is case insensitive, unlike LOOKUP. Partial matches return False.

Note that under certain (unlikely) conditions EXISTS cannot operate correctly. It will fail if you specify a complex formula that uses both LOOKUP and EXISTS on the same file, or EXISTS twice with different values.

See Also

LOOKUP

EXP

Purpose

Returns the value of the mathematical constant 'e' raised to a power.

Syntax

EXP (*nexpr*)

Comments

EXP gives the value of 'e' to the power of *nexpr*.

nexpr has a maximum absolute value of 709.7827128934, and a range of -709.7827128934 to +709.7827128934. This in turn determines that the largest numeric value that Superbase can hold is 1.797693134862 times 10 to the power 308.

Examples

- 1 numfieldc = EXP(numfielda)
- 2 numfieldc = EXP(datefielda - datefieldb)
- 3 IF numfieldc > EXP(numfielda * numfieldb) THEN ...
- 4 x#% = EXP(y#%)
- 5 x#% = EXP(y#% * numfielda * (datefielda - datefieldb))
- 6 x#% = EXP(VAL(RIGHT\$(textfielda, 5)))
- 7 ? EXP(x#%)

See Also

LOG

EXPORT

Purpose

Exports to an external file from the current file.

Syntax

```
EXPORT[ FILE sbfname][ INDEX index][ DESCENDING][ TO] filename  
[ WHERE conditions/ASK][ USING params/FILE/"0"/"2"/"3"/LABELS]
```

Comments

This command is the program equivalent of the EXPORT option in the PROCESS menu. It converts a Superbase file to a specified format and stores it on disk.

The *filename* parameter gives the name under which the file will be stored after it has been exported. It is this parameter that determines the file type.

To export a file in dBase or Enable format, you must use the DBF extension name. You must also specify whether it is to take the format for dBase II or dBase III by following the EXPORT command with the USING parameter. USING "2" exports in dBase II format, USING "3" exports in dBase III format. For example:

```
EXPORT "address.dbf" USING "3"
```

This converts the current Superbase database file to dBase III format.

You may also export Enable files. If you wish to create an Enable file definition (SBF extension), use the parameter "0" instead of "2" or "3."

To convert a file to a spreadsheet format, you must give it one of the following extension names:

.xls	(for Excel files)
.wks	(for Lotus files)
.lgx	(for Logistix files)
.spp	(for Superplan files)
.dif	(for DIF files)

If you specify the USING LABELS option for spreadsheet files, the field names in the Superbase file will be exported as a row of labels.

To export a file in ASCII format, choose a different extension name from the ones listed above. Alternatively, you can omit the extension name for ASCII files.

Details of the syntax for exporting files in ASCII format are given at the end of this section. The next two parameters are used with all file types. FILE *sbfname* specifies the Superbase file which is to be converted. If this parameter is not given, EXPORT takes data from the current open file.

WHERE *conditions* allows you to create a filter to determine which records are copied to the export. *conditions* is set up in the same way as the command string in the Filter requester. For an explanation of how the ASK parameter is used, see the entry for SELECT WHERE.

If INDEX *indexname* is not specified, the command exports records in the order of the current index. Use INDEX to specify a different order. The default order for exporting is ascending. Include the DESCENDING keyword if you require the exported output in descending order. This parameter is not available if you are exporting a spreadsheet file.

ASCII files

There are two ASCII file formats, ASCII delimited and ASCII fixed length. To select the ASCII fixed length format, you must enter the USING FILE parameter at the end of the command.

For ASCII delimited files, you may select the USING *parameters* option. This allows you to change the Export/Import parameters as specified in the Options requester on the Set menu. These determine which characters Superbase uses to separate fields and records in the ASCII file.

You can also specify whether text fields are exported with or without quotation marks around them. If you do not specify USING *parameters*, Superbase will use the parameters which are set in the Options requester.

USING takes three parameters, each enclosed by quotation marks and separated by commas. For example:

```
USING "&","$$","0"
```

The first two parameters specify the field separation characters and the record separation characters. As in the Options requester, you can define a separator using one character or two characters. The third parameter must be either 0 or 1; 0 for no quotation marks, 1 to include quotation marks.

If you want to use a non-printing character – the carriage return character or the line feed character, for example – as a separator, you need to enter its ASCII code with the CHR\$ function.

For example:

```
USING "& ",CHR$(13)+CHR$(10),"0"
```

defines the record separator as the carriage return character followed by the line feed character.

Examples

```
1 EXPORT "aaa.dbf" USING "2"
```

Exports the current file converting it to the dBase file 'aaa.dbf' in the format for dBase II.

- 2 EXPORT FILE "aaa" INDEX fielda TO "aaa.wks" WHERE
(datefield) < DAYS("29 Apr 87")

Exports records in the file 'aaa' whose date field value is less than 29th April 1987. Stores them in Lotus file format with the name 'aaa.wks'.

- 3 EXPORT FILE "aaa" TO "aaa.exp" USING "&","##","1"

Exports the data in the file 'aaa' to ASCII delimited file 'aaa.exp', using the separators specified.

- 4 EXPORT "aaa.exp" WHERE ASK USING FILE

Exports the current file to the ASCII fixed length file 'aaa.exp'. When the command is executed, ASK brings up a filter requester where the user can enter the export filter conditions.

See Also

COPY, IMPORT, REORGANIZE, SELECT TO FILE

FCASE\$

Purpose

Converts the first letter of a text string to upper case, converting the rest of the string to lower case.

Syntax

FCASE\$ (*stexpr*)

Comments

FCASE\$ takes a word and makes the first letter a capital letter.

Associated functions are LCASE\$ and UCASE\$.

Examples

- 1 textfielda = FCASE\$(textfielda)
- 2 x\$ = FCASE\$(y\$)
- 3 x\$ = FCASE\$("ABCDEF")
- 4 ? FCASE\$(x\$)

See Also

LCASE\$, UCASE\$

FILE

Purpose

Makes an open file the current file or returns the name of the current file.

Syntax

FILE [*sbfname*]

Comments

When several files have been opened, you can use FILE to make one of them the current file.

FILE also acts as a system variable which holds the name of the current database file.

Examples

```
1  FILE "aaa"
2  x$ = "bbb": FILE x$
3  a$ = "address": b$ = "bank"
   OPEN FILE a$: SELECT FIRST
   OPEN FILE b$: SELECT FIRST
   PAGING OFF
   f$ = b$
   lab1: VIEW: WAIT x$
   IF EOF(a$) OR EOF(b$) THEN END
   IF f$ = b$ THEN f$ = a$ ELSE f$ = b$
   FILE f$: SELECT NEXT
   GOTO lab1
4  IF FILE = "Address" THEN ...
5  a$ = FILE
```

Notes

The third example shows how you can display records from two (or more) files at the same time. It displays one record after another, alternating between two files until it reaches the end of the shorter file. Examples 4 and 5 demonstrate the use of FILE as a system variable.

See Also

? FILE, INDEX, OPEN FILE

FIX

Purpose

Sets the accuracy with which DML stores a real number and performs calculations on it, and also performs rounding.

Syntax

FIX (*nexpr1* , *nexpr2*)

Comments

FIX allows you to limit a numeric expression to a specified number of decimal places. *nexpr2* is the number of decimal places that *nexpr1* is evaluated at.

Superbase stores real numeric variables at 14 figure accuracy, so the fraction 1/3 is stored as 0.33333333333333. Adding two of these together gives 0.666666666666667. If you set the numeric format to two decimal places in a file definition, Superbase would show this result as: $0.33 + 0.33 = 0.67$.

This is not incorrect, especially if you are a scientist or engineer, but it is not very helpful if you are using a Query to produce an invoice. `FIX (1/3,2)` stores the fraction 1/3 as 0.330000000000, so `FIX(1/3,2) + FIX(1/3,2)` gives the result as 0.66.

However, `FIX` also rounds numbers, so `FIX(2/3,2)` stores the fraction 2/3 as 0.670000000000.

Note that numeric fields with two decimal places are automatically `FIXed` (i.e., they are rounded to two decimal places). You may also use the output format parameter `&` to set the number of decimal places.

Examples

- 1 `numfieldc = FIX(numfielda, 3)`
- 2 `numfieldc = FIX(datefielda - datefieldb, 0)`
- 3 `IF numfieldc > FIX(numfielda * numfieldb, 4) THEN ...`
- 4 `x#% = FIX(y#%, 2)`
- 5 `x#% = FIX(y#% * numfielda * (datefielda - datefieldb), 2)`
- 6 `x#% = FIX(VAL(RIGHT$(textfielda, 5)), 2)`
- 7 `? FIX(x#%, 2)`

See Also

`INT`

FN alpha

Purpose

Returns a string containing only alphabetic characters (a-z, A-Z).

Syntax

FN alpha(*strexpr*)

Comments

The function is used to strip all non-alphabetic characters out of a string.

Examples

```
1  x$ = "abcd-123;"  
   y$ = FN alpha (x$)
```

The string y\$ contains 'abcd'.

See Also

FN NUMERIC

FN ansi

Purpose

Returns a string in which all IBM characters are converted to the ANSI character set.

Syntax

FN ansi(*strexpr*)

Comments

The function converts characters in the IBM/OEM character set to their ANSI equivalents. See also FN ibm.

Examples

```
1  x$ = FN ansi(y$)
```

See Also

FN IBM

FN dec

Purpose

Returns the decimal equivalent of a hexadecimal number

Syntax

FN dec (*strexpr*)

Comments

The string expression should contain hexadecimal digits.

Examples

```
1  x$ = "1F"  
   y&% = FN dec (x$)
```

The variable y&% has the value 31.

See Also

FN HEX

FN ext

Purpose

Returns a string containing the extension of a filename.

Syntax

FN ext (*strexpr*)

Comments

The string expression should be a filename together with its path.

Examples

```
1  x$ = "dh0:\data\myfile.sbf"  
   y$ = FN ext (x$)
```

The string y\$ contains '.sbf'.

See Also

FN NAME, FN PATH, FN ROOT

FN fact

Purpose

Returns the factorial of a number.

Syntax

FN fact(*nexpr*)

Comments

The factorial of *nexpr* is equal to:

$(nexpr) * (nexpr-1) * (nexpr-2) * \dots * (nexpr-n)$

Where $nexpr-n = 1$. If *nexpr* is not an integer, it is truncated.

Examples

1 x&% = FN fact(5)

The result would be 5 times 4 times 3 times 2 times 1, which is 120.

FN fv

Purpose

Returns the future value of an investment, for example the result of saving a fixed amount each month for a number of years.

Syntax

FN fv(*rate*, *nper*, *pmt* [, *pv* [, *type*]])

Comments

The arguments for FN fv must be numeric; they may be numbers, variables or fields. *rate* is a constant interest rate. *nper* is a number of periods. *pmt* is a constant payment amount. *pv* is the present value of an investment. *type* may be either 0 or 1; 0 computes for payments at end of month, 1 for beginning.

Examples

1 FN fv(10, 12, -1000)

Notes

The optional *p_v* and *type* arguments are assumed to be zero if omitted. Outgoing amounts should be supplied as negative numbers.

See Also

FN NPV, FN PMT, FN PV, FN RATE, FN SLN

FN hex**Purpose**

Returns a string containing the hexadecimal equivalent of a decimal number.

Syntax

FN hex (*nexpr*)

Comments

The numeric expression is evaluated, and truncated if necessary, to give an integer value.

Examples

```
1  x%% = 31
   y$ = FN hex (x%%)
```

The string y\$ contains '1F'.

See Also

FN DEC

FN ibm

Purpose

Returns a string in which all ANSI characters are converted to the IBM character set.

Syntax

FN ibm(*strexpr*)

Comments

The function converts characters in the ANSI character set to their IBM/OEM equivalents. See also FN ansi.

Examples

```
1  x$ = FN ibm(y$)
```

See Also

FN ANSI

FN name

Purpose

Returns a string containing a filename together with its extension.

Syntax

FN name (*strexpr*)

Comments

The string *strexpr* should be a filename together with its path.

Examples

```
1  x$ = "dh0:\data\myfile.sbf"
   y$ = FN name (x$)
```

The string y\$ contains 'myfile.sbf'.

See Also

FN EXT, FN PATH, FN ROOT

FN nper

Purpose

Returns the number of payments on an investment. For example, 11 or more monthly payments of \$100 may be needed to repay a loan of \$1000, depending on the interest rate.

Syntax

FN nper(*rate*,*pmt*,*pv*[,*fv*[,*type*]])

Comments

The arguments for FN nper must be numeric; they may be numbers, variables or fields.

rate is a constant interest rate. *pmt* is a constant payment amount. *pv* is the present value of an investment. *fv* is the future value of an investment. *type* may be either 0 or 1; 0 computes for payments at end of month, 1 for beginning.

Examples

1 FN nper(.5/12, -100, -1000)

Notes

The optional *fv* and *type* arguments are assumed to be zero if omitted. Outgoing amounts should be supplied as negative numbers.

See Also

FN FV, FN PMT, FN PV, FN RATE, FN SLN

FN numeric

Purpose

Returns a string containing only numeric characters (0-9).

Syntax

FN numeric(*strexpr*)

Comments

The function is used to strip all non-numeric characters out of a string.

Examples

```
1  x$ = "abcd-123;"  
   x$ = FN numeric (x$)
```

The string y\$ contains '123'.

See Also

FN ALPHA

FN path

Purpose

Returns a string containing all pathname characters including final backslash.

Syntax

FN path (*strexpr*)

Comments

The string expression should contain a filename together with its path.

Examples

```
1  x$ = "dh0:data/myfile.sbf"  
   y$ = FN path (x$)
```

The string y\$ contains 'dh0:data/'.

See Also

FN EXT, FN NAME, FN ROOT

FN pmt

Purpose

Returns the value of a periodic payment, such a loan repayment.

Syntax

FN pmt(*rate*, *nper*, *p*, *fv*, *type*)

Comments

The arguments for FN pmt must be numeric; they may be numbers, variables or fields. *rate* is a constant interest rate. *nper* is a number of periods. *p* is the present value of an investment, for example the total value to the lender of the

payments on a loan. *fv* is the future value of an investment. *type* may be either 0 or 1; 0 computes for payments at end of month, 1 for beginning.

Examples

1 FN pmt(.8/12, 12, 0)

Notes

The optional *fv* and *type* arguments are assumed to be zero if omitted. Outgoing amounts should be supplied as negative numbers.

See Also

FN FV, FN NPER, FN PV, FN RATE, FN SLN

FN pv

Purpose

Returns the present value of an investment, for example the total value of the payments on a loan to the lender.

Syntax

FN pv(*rate*,*nper*,*pmt*[,*fv*[,*type*]])

Comments

The arguments for FN pv must be numeric; they may be numbers, variables or fields. *rate* is a constant interest rate. *nper* is a number of periods. *pmt* is a constant payment amount. *type* may be either 0 or 1; 0 computes for payments at end of month, 1 for beginning.

Examples

1 FN pv(.5, 60, 250)

Notes

The optional *fv* and *type* arguments are assumed to be zero if omitted. Outgoing amounts should be supplied as negative numbers.

See Also

FN FV, FN NPER, FN PMT, FN RATE, FN SLN

FN rate

Purpose

Returns the interest rate per period for an investment.

Syntax

FN rate(*nper*, *pmt*, *pv* [, *fv* [, *type* [, *guess*]]])

Comments

The arguments for FN rate must be numeric; they may be numbers, variables or fields.

nper is a number of periods. *pmt* is a constant payment amount. *pv* is the present value of an investment. *fv* is the future value of an investment. *type* may be either 0 or 1; 0 computes for payments at end of month, 1 for beginning. *guess* is a decimal estimate of the rate, assumed to be 0.1 if omitted.

FN rate attempts to derive a result near to *guess*. If it cannot do so after a number of iterations, it stops with the message 'Formula Too Complex'.

Examples

1 FN rate(60, -200, 10000)

Works out the monthly rate on a five year loan of \$10,000 repaid at \$200 a month.

Notes

The optional *fv* and *type* arguments are assumed to be zero if omitted. Outgoing amounts should be supplied as negative numbers.

See Also

FN FV, FN NPER, FN PMT, FN PV, FN SLN

FN root

Purpose

Returns a string containing the root of a filename.

Syntax

FN root (*strexpr*)

Comments

The string expression should be a filename together with its path.

Examples

```
1  x$ = "dh0:data/myfile.sbf"  
   y$ = FN root (x$)
```

The string y\$ contains 'myfile'.

See Also

FN EXT, FN NAME, FN PATH

FN sln**Purpose**

Returns straight line depreciation for an asset for a single period.

Syntax

FN sln(*cost, salvage, life*)

Comments

The arguments for FN sln must be numeric; they may be numbers, variables or fields. *Cost* is the initial cost of the asset. *Salvage* is the value of the asset at the end of the depreciation period. *Life* is the number of periods, usually years, over which the depreciation is being calculated.

Examples

```
1  x#% = FN sln(3000, 750, 10)
```

An asset bought for \$3000 and worth \$750 dollars after 10 years depreciates by \$225 in the first year.

See Also

FN FV, FN NPER, FN PMT, FN PV, FN RATE

FN sys

Purpose

Returns system information.

Syntax

FN sys(*nexpr*)

Comments

FN sys takes a numeric argument from the following list. The value returned is also numeric.

Argument	Returns
0	Workbench Release number, e.g. 13 or 20 (= Release 1.3 or 2.0)
1	Workbench Major Version number, e.g. 34 or 36
2	Superbase Version number, e.g. 16 (= hex 10 = Version 1.0)
3	Superbase internal release number
4	Superbase product type (the sum of whichever of the following apply: 1=Network, 2=Runtime, 4=Demo, 8=Superbase 2, 128=Amiga)
5	Superbase user count (applies to Lan product only)
6	Screen width in pixels
7	Screen height in pixels (scan lines)
8	Number of planes
9	x co-ordinate of Superbase main window
10	y co-ordinate of Superbase main window
11	Superbase main window width in pixels
12	Superbase main window height in pixels (scan lines)
13	Printer logical pixels per inch in x direction
14	Printer logical pixels per inch in y direction
15	Current printable page size in x direction in mm
16	Current printable page size in y direction in mm

Examples

```

1  x%% = FN sys(5)
   IF x% > 6 THEN
       REQUEST "Max # Licensed Users Exceeded:",
           "Application Terminating",139
       QUIT
   END IF

```


FOOTING

Purpose

Sets the Report footing for every page.

Syntax

FOOTING *nexpr*

Comments

FOOTING is followed by a number of ? statements specifying the information that will appear at the bottom of every page in a report. These must be followed by an END FOOTING statement.

nexpr specifies the number of lines required for the report footing. Make sure that the number of lines output by your ? statements is the same as the number specified by *nexpr*.

Examples

```
1 FOOTING 1
  ? "Page ";PG
  END FOOTING
```

Notes

This example uses the system variable PG to print the page number. See the entry HEADING for more details.

See Also

END FOOTING, HEADING, REPORT

FOR TO NEXT

Purpose

Repeats a series of program statements a specified number of times.

Syntax

FOR *var* = *nexpr1* TO *nexpr2* [STEP *nexpr3*] *statements* NEXT [*var*] [,*var*]

Comments

This command sets up a program loop in which the statements between FOR TO and NEXT are executed a given number of times, using *var* as a counter. *nexpr1* sets the initial value of the counter and *nexpr2* sets the final value.

If STEP *nexpr3* is not included, the counter is increased by one every time the program executes the statements inside the loop; that is, every time it passes from FOR to NEXT. When the counter reaches the value set in *nexpr3*, the program moves on to the statement after NEXT.

When STEP *nexpr3* is included, the program increases the counter by the amount specified in *nexpr3*.

var must be a numeric variable; *nexpr1*, *nexpr2*, *nexpr3* can be any numeric expression, including other numeric variables.

You can use FOR TO NEXT on the same line with multiple statements, or on multiple lines. But if there is more than one statement between FOR and NEXT, it is advisable to put each on a separate line.

If a number of FOR NEXT loops end at the same point, you can use a single NEXT statement for all of them.

```
NEXT n%%, y&%, z#%
```

is the same as

```
NEXT n%%
NEXT y&%
NEXT z#%
```

When counter variable names are specified in NEXT statements, inner loops must be terminated before outer loops. If NEXT is not followed by the name of a counter variable, it forms the end of only the most recently begun loop.

Examples

- 1 FOR i#% = 10 TO 1 STEP -0.5: ? i#% , i#% ^ 2: NEXT
- 2 FOR n%% = 1 TO 200
? n%, ERR\$(n%)
NEXT
- 3 weekcount%% = 1
FOR i&% = DAYS("01/01/91") TO DAYS("31/03/91") STEP 7
? "Week"; &2.0 weekcount%%
FOR j%% = 0 TO 6: k&% = i&% + j%%
? &2.0 DAY(k&); " "; MONTH\$(k&), DAY\$(k&)
NEXT j%%
weekcount%% = weekcount%% + 1
NEXT i&%

Notes

Example 2 displays the Superbase error messages for error numbers 1 to 200.
Example 3, which prints out the weekdays for the first quarter of 1991 in weeks, shows how FOR TO NEXT statements may be nested.

See Also

WHILE ... WEND

FORM

Purpose

Specifies which part of a Form is displayed in the database window, or returns the name of the current form.

Syntax

FORM [SHOW] [*page* [,*row*,*col*]]

Comments

This command is used to select a particular page within a multi-page form or to bring part of a page that is outside the database window into view.

page must be a numeric expression specifying the page number. *row* and *col* must be numeric expressions which give a row and column position within the page. Form will then move the page so that the specified position is at the top left-hand corner of the database window.

The optional parameter SHOW is used to display external files. If the form contains external file fields, FORM SHOW is equivalent to clicking on the camera button.

When the FORM command is used without any following parameters, Superbase re-displays only the field data in a form, leaving the remainder of the display unchanged.

When FORM is used as a system variable, it returns the name of the current form. If no form is opened, a null string will be returned.

See Also

See also OPEN FORM.

Examples

1 FORM 2

Selects page 2 in the Form currently displayed.

2 FORM 1,24,1

Makes row 24, column 1 of page 1 appear at the top left-hand corner of the database window.

3 FORM SHOW

Displays any external files associated with the form.

4 FORM

Continues to display the current form, but redraws all field data so that any updated information is displayed.

5 x\$ = FORM

Stores the name of the current open form in x\$.

See Also

VIEW, SHOW

FORMAT\$

Purpose

Forces query and report output to wordwrap within a column.

Syntax

FORMAT\$ (*stexpr*, *nexpr1* [,*nexpr2*])

Comments

FORMAT\$ formats a text field or variable to a specified width by inserting carriage return characters.

Stexpr is normally the fieldname of a text field, which may be up to 4000 characters long (see Note below). *Nexpr1* defines the width of the column in characters.

Nexpr2 specifies the line of the field to be output, allowing you to position formatted text precisely. If *nexpr2* is omitted, the formatted text will be output in its entirety before any other items are output.

Examples

```
1  SELECT name.CLIENTS,  
   @30 FORMAT$(comments.CLIENTS,40,1),telephone.CLIENTS,  
   @30 FORMAT$(comments.CLIENTS,40,2),  
   @30 FORMAT$(comments.CLIENTS,40,3),  
   @30 FORMAT$(comments.CLIENTS,40,4),  
   @30 FORMAT$(comments.CLIENTS,40,5)  
  
2  ? FORMAT$(field1,25)
```

Example 1 prints name, first line of comments field, and telephone number, then prints the next four lines of comments. The text in the comments column will be wordwrapped. Example 2 prints the contents of field1 in a column 25 characters wide, wordwrapping text when necessary.

Notes

Because FORMAT\$ actually inserts formatting information into a field or variable, the formatted length may exceed 4000 characters. In this case, characters beyond the 4000 maximum will be discarded.

FOUND

Purpose

Detects whether a key lookup has been successful or not.

Syntax

FOUND (*strexpr*)

Comments

After you have issued a SELECT KEY command to search for a particular record, FOUND will tell you whether the record has been found. If the search is successful, FOUND returns a value of -1 (true); if the search is unsuccessful, FOUND returns 0.

strexpr should contain the file name of an SBF file which has already been opened. However, you can use the empty string – as in FOUND ("") – to refer to the current file.

FOUND is only set and cleared by the SELECT KEY and LOOKUP commands.

Examples

```
1  SELECT KEY "London"  
   ? FOUND ("" )
```

```
2  a$ = "Clients": SELECT KEY "Smith" FILE a$  
   IF FOUND (a$) THEN ? "Smith found" ELSE ? "No Smiths"
```

See Also

SELECT KEY

FREE

Purpose

Returns the amount of free memory.

Syntax

FREE (*nexpr*)

Comments

This function returns a number showing how much free memory is available.

nexpr determines whether the number refers to chip, contiguous or fast memory.

0 = total free memory

2 = chip memory

4 = fast memory. Adding 2^{17} to one of these parameters gives the largest block of memory available in its respective memory area. So, FREE ($2^{17} + 2$) returns the maximum area of contiguous chip memory.

Examples

```
1  ? FREE(0)
```

```
2  f&% = FREE(0)
```

GET

Purpose

Gets a character from the keyboard.

Syntax

GET *strvar* / *field*

Comments

This command reads a character from the keyboard into a string variable or a text field. It does not wait for a keystroke, and if no key is pressed it returns an empty string. If you want GET to wait until a key is pressed, you need to place it in a loop, as in:

```
LAB1: GET A$: IF A$="" GOTO LAB1
```

Alternatively, use the WAIT command.

Note that control characters can be returned in a GET statement. If these are saved in a file which is subsequently LISTed, they will generate the 'File contains non-text characters' error. Note also that CTRL+C, the standard interrupt key, does not stop program execution if returned in GET or WAIT. It is up to the program to detect the value and act appropriately.

Examples

```
1 Loop: GET a$  
  IF a$ = "" THEN GOTO LOOP ELSE ? a$
```

Notes

If you want GET to return the space character, set BREAK OFF beforehand. Otherwise, pressing the space bar will pause the program.

See Also

ASK, WAIT

GOSUB

Purpose

Calls a procedure or subroutine.

Syntax

GOSUB *label*

Comments

Like the GOTO statement, GOSUB transfers control to a different part of the program: it causes the program to branch to the label specified. But unlike GOTO, it remembers where it branched from. When the program meets a RETURN statement, it jumps back to the line following the GOSUB statement.

GOSUB is used to call a subroutine; that is, one or more program lines which perform a specific task and can be called from different places within the main program. Subroutines are useful if the same task needs to be performed at several different stages in the program. Instead of repeating a group of lines, it saves space and is more convenient to put them in a subroutine.

In DML, a subroutine is defined by a label at the beginning and a RETURN statement at the end.

Examples

```
1  GOSUB sub1
2  x$ = Address(1): GOSUB label1
   x$ = Address(2): GOSUB label1
   ....
   ....
   ....
   label1:
   ? "This subroutine outputs x$ to the printer"
   PRINT x$
   DISPLAY;
   RETURN
```

See Also

CALL, GOTO, ON GOSUB, RETURN

GOTO

Purpose

Transfers control to another part of the program.

Syntax

GOTO label

Comments

This statement makes the program jump to the label specified, instead of continuing to the next line in the program. It alters the order in which DML executes a program. This kind of control transfer is called an unconditional jump. For conditional jumps, see the ON GOTO and the IF THEN ELSE statement.

Examples

```
1  GOTO fred
   ....
   ....
   fred: ....
```

See Also

GOSUB, ON GOTO

GROUP

Purpose

Specifies the field on which a report is grouped and the field(s) for which subtotal reporting is required.

Syntax

GROUP *fieldname* [*fieldname*] [,...]

Comments

The Superbase Form Designer inserts a GROUP statement in a report program when you specify fields with the Group command on its Report Generator menu. GROUP has two main functions. First, it defines the field on which Superbase groups data in a report. Second, it specifies any other fields for which subtotals – or other reporting features such as MAX and MEAN – are required.

If you wish to specify several levels of grouping, you should enter a separate GROUP statement for each level.

GROUP can also be used in an AFTER GROUP section as a reference for the field which defines the group. Since the group has already changed, entering the field name would output the data for the next group; GROUP allows you to retrieve the data for the previous group – the group for which the AFTER GROUP section provides reporting information such as subtotals and record counts.

Examples

1 GROUP City, amount

'City' is the field on which the report is grouped, 'amount' is a field for which subtotals are required – the AFTER GROUP section should include the statement:
SUM amount

2 GROUP Country
GROUP City, amount

In this example, record data is grouped at two levels: City within Country.

See Also

AFTER GROUP, BEFORE GROUP, COUNT, END GROUP, MAX, MIN, REPORT, SD, SUM, VAR

HEADING

Purpose

Sets the Report heading for each page.

Syntax

HEADING

Comments

Superbase generates a HEADING statement when you specify a report heading using the Heading option on the Form Designer's Report menu. This statement marks the start of a HEADING section and is followed by one or more statements which define the heading for a report. The section must end with an END HEADING statement.

If you wish to include the page number in a heading, use the system variable PG, as in:

```
? @32;"Page ";PG
```

PG is set to one when the Report Select command is executed and incremented by one after each page.

Examples

```
1  HEADING  
   ? @22; BF; UL; "DEPOSITS REPORT on", TODAY, "at", NOW;  
   ATTR OFF  
   END HEADING
```

See Also

END HEADING, FOOTING, REPORT

HOME

Purpose

Takes the cursor to the top left-hand corner of the screen.

Syntax

HOME

Comments

HOME moves the cursor to the top of the screen without clearing the screen. The next screen output will now appear at that position.

HOME is useful when you want to overwrite something you have previously displayed on screen, or when you want to move the cursor up the screen or backwards. Normally, if you attempt to do this with LOCATE or using the @ parameter with the ? command, DML displays the next page. HOME allows you to display text and numbers anywhere on screen, no matter what has been displayed previously.

Examples

```
1  CLS
   FOR r%% = 1 to 18
   FOR c%% = 1 to 80
   HOME : LOCATE c%%, r%%: ? " Hello "
   NEXT
   NEXT
```

See Also

? @, COL, CLS, LOCATE, ROW

HRS

Purpose

Extracts the number of hours from a numeric expression containing the time in thousandths of a second.

Syntax

HRS (*nexpr*)

Comments

nexpr will usually be a time field or the result of a TIMEVAL expression.

Examples

```
1  hr%% = HRS(timefield)
```

See Also

MINS, SECS, THOUSECS

IF THEN ELSE

Purpose

Executes a statement if a condition is true.

Syntax

IF *expr* THEN *statements* [ELSE IF *expr* THEN *statement*] [END IF] [ELSE *statements*] [END IF]

Comments

expr can be any expression – string, numeric or logical – which is capable of being true or false. For example, A\$ = "Smith" is either true or false, so:

```
IF A$="Smith" THEN ...
```

is a valid statement, but

```
IF A$ THEN ...
```

is not.

If *expr* is true, DML executes the statement or statements after THEN. Otherwise it proceeds to the next statement after the IF THEN statement.

THEN can be omitted when it is followed by GOTO. For example:

```
IF expr GOTO label
```

is the same as

```
IF expr THEN GOTO label
```

By including the ELSE option, you can instruct DML to choose between two courses of action. It executes the statements after THEN if *expr* is true; if *expr* is false, it executes the statements after ELSE.

An IF THEN ELSE statement can be split up so that it is placed on several lines. If you do this, the statements after THEN must start on a new line, and ELSE must also start on a new line. For example:

```
IF x## > y## THEN ? "Greater than" ELSE ? "Less than or  
equal"
```

can be written as:

```
IF x## > y## THEN  
? "Greater than"  
ELSE  
? "Less than or equal"  
END IF
```

END IF marks the end of the IF THEN ELSE statement. With single line IF THEN statements it is optional, but it must be placed at the end of a block IF THEN

statement. Thus it should always be used when ELSE is placed on a separate line. If it is not used, DML assumes that all the separate line statements in the rest of the program belong to ELSE.

END IF should also be used when the ELSE option has not been selected, but the statements following THEN are placed on separate lines. For example:

```
IF b$ = "Yes" THEN n%% = 1: z%% = 2: GOSUB label
```

can be written:

```
IF b$ = "Yes" THEN
n%% = 1
z%% = 2
GOSUB label
END IF
```

.....

.....

Here too, END IF is used to tell DML where the statements belonging to THEN finish.

Within a block IF THEN statement, you can insert one or more ELSE IF statements. These are used to extend the number of alternatives provided by the original block IF THEN; but they do not require additional END IF statements.

expr does not always need to contain an operator. Remember that DML assigns a value of 0 to false expressions and -1 to true expressions. With IF THEN commands, though, DML treats a numeric expression with any value other than zero as true. And it treats a numeric expression with any value other than -1 as false.

```
IF EOF("") THEN .....
```

implies:

```
IF EOF("") <> 0 THEN ...
```

Likewise,

```
IF NOT EOF("") THEN ...
```

implies:

```
IF EOF("") <> -1 THEN
```

Examples

- 1 IF *expr* THEN *x*&% = *x*&% + 1
- 2 IF *expr* THEN *x*\$ = "TRUE"
- 3 IF *expr* THEN *x*&% = *x*&% + 1: *x*\$ = "TRUE"
- 4 IF *expr* THEN *labltrue*

```
5  IF expr THEN GOTO labltrue
6  IF expr THEN x$ = "TRUE" ELSE x$="FALSE"
7  IF expr THEN x$ = "TRUE": x%% = x%% + 1 ELSE x$="FALSE"
8  IF b$ = MID$(a$,3,1) THEN
    x%% = 1
    GOSUB label1
  ELSE
    x%% = 2
    GOSUB label2
  END IF
9  IF a$ = "Y" or a$ = "y" THEN
    ? "Yes"
  ELSE
    IF a$ = "N" or a$ = "n" THEN
      ? "No"
    ELSE
      ? "Other"
    END IF
  END IF
10 IF a% THEN
    ? atrue$
    IF b% THEN
      ? btrue$
    ELSE
      ? bfalse$
    END IF
  ELSE
    ? afalse$
  END IF
11. IF a% = 1 THEN
    ? "one"
  ELSE IF a% = 2 THEN
    ? "two"
  ELSE IF a% = 3 THEN
    ? "three"
  ELSE
    ? "Not 1 to 3"
  END IF
```

Notes

Examples 9 and 10 show how IF THEN statements can be nested. In both cases, the second IF THEN statement is only executed if the condition in the first

statement gives a false result. Example 11 illustrates the use of the ELSE IF statement.

See also the SELECT CASE statement.

See Also

FOR ... NEXT, SELECT CASE, WHILE ... WEND

IMPORT

Purpose

Converts dBase, Enable, ASCII, and spreadsheet files into Superbase files, merges an ASCII file into an open Superbase file.

Syntax

The syntax for this command varies according to type of file being converted or merged.

ASCII files

IMPORT *filename* [TO NEW FILE / TO FILE*sbfname*] [WHERE *conditions* / ASK] [USING *parameters*] / [USING FILE]

dBase and Enable files

IMPORT *filename.dbf*

Spreadsheet files

IMPORT *filename.lgx/.wks/.wk1/.dif/.spp* [USING *strexpr1*, *strexpr2* [,*strexpr3*]]

Superbase files

IMPORT *filename.sbf* [[TO] FILE*sbfname*] [WHERE *conditions* / ASK]

Comments

The extension name given as part of the *filename* parameter specifies the type of file that is to be read from disk. All the types of input file described below may be converted by IMPORT into Superbase database files; certain types of input file may also be merged into existing Superbase database files.

dBase and Enable files

To convert a dBase file, you must supply the DBF extension – no further parameters are required. The file will be converted to a Superbase database file with an index on the first field.

Spreadsheet files

To convert a spreadsheet file, use one of the following extensions:

- .spp (for Superplan files)
- .xls (for Excel files)

.lgx (for Logistix files)
.wks or .wk1 (for Lotus files)
.dif (for DIF files)

The first two parameters following USING specify the range of cells that is to be converted. *strexpr1* gives the address of the cell in the top left-hand corner, *strexpr2* gives the address of the bottom right-hand cell. If the USING parameters are not entered, Superbase converts the entire spreadsheet file.

With *strexpr3*, you can give the number of a row in the spreadsheet data which contain labels. These will then be taken as the Superbase file's field names. If you leave this parameter out, the field names will be formed by taking the spreadsheet's column identifiers and adding an underscore character to them; i.e, data in column AB will be stored in field AB_.

ASCII files

With ASCII files, the data is imported into the current file or another open file named with the TO FILE parameter. To indicate an ASCII file you can use any (or no) extension name except those listed above.

Superbase can import two types of ASCII file, ASCII fixed length and ASCII delimited. The USING FILE parameter indicates that the file to be imported is ASCII fixed length. If this is omitted, it is assumed that the file is ASCII delimited.

The NEW option indicates that a new Superbase file is to be created, as for a spreadsheet. If NEW is omitted, the ASCII file is imported into the named file, or the current file if there is no named file. Only ASCII delimited files may be imported using NEW.

With ASCII delimited files, USING *parameters* can be used to change the default Import/Export values set in the Options requester – i.e. to indicate which characters are used for field and record separators, and whether the field data is enclosed in quotation marks or not. For a description of USING and the parameters it takes, see EXPORT.

With both ASCII file types, WHERE *conditions* allows the user to add a filter to the IMPORT command. *condition* is set up in the same way as the command line in the Filter requester. For details of the ASK parameter, see SELECT WHERE.

Superbase Files

You can use this command to merge one Superbase file into another. The SBF extension is compulsory. The files need not have the same structure, as a selected fields list may be defined. See the DML entry for OPEN FIELDS.

The WHERE *conditions* option allows the user to add a filter to the IMPORT command. *condition* is set up in the same way as the command line in the Filter requester. The source file must be opened prior to the import if a filter is used. For details of the ASK parameter, see SELECT WHERE.

If the TO *sbfname* option is omitted, the named file will be merged into the current file.

Examples

1 IMPORT "aaa.exp"

Imports the ASCII delimited file "aaa.exp" into the current open file.

2 IMPORT "aaa.exp" TO FILE "aaa"
 WHERE (datefield) > DAYS ("Apr 29 87") USING FILE

Imports the ASCII fixed length file "aaa.exp" into the Superbase file "aaa", taking only the records whose date field value is later than April 29th, 1987.

3 IMPORT "aaa.exp" WHERE ASK USING CHR\$(44), CHR\$(13), "0"

Imports the ASCII delimited file 'aaa.exp' into the current open file, and specifies that the file uses the comma and carriage return characters as field and record separators. It also indicates that the field data is not enclosed in quotation marks. The WHERE ASK parameter brings up a filter requester when the command is executed.

4 IMPORT "aaa.dbf"

Converts the dBase file "aaa.dbf" into a Superbase file named "aaa.sbf."

5 IMPORT "aaa.lgx"

Converts the Logistix file 'aaa.lgx' into a Superbase file named 'lgx'.

6 IMPORT "aaa.wks" USING "a1", "k50", "1"

Converts the Lotus file 'aaa.wks' into a Superbase file named 'aaa.sbf'. The USING parameters specify that only the cells in the range a1 to k50 are converted, and that the labels in row 1 are used as field names. Assuming that every row and column within the range is occupied, the Superbase file will have 50 records in it, with 11 fields to a record.

7 IMPORT "newdata.sbf" TO "olddata"
 WHERE datefield.Newdata >= "Jan 1 89"
 AND datefield.Newdata <= "Jan 31 89"

Merges records from 'Newdata' into 'Olddata' provided that the date in datefield is in January 1989.

8 IMPORT "newdata.asc" TO NEW FILE "newdata"

Generates the Superbase file 'newdata.sbf' from 'newdata.asc', which is an ASCII disk file.

See Also

EXPORT, INPUT, OPEN ... FOR INPUT

INDEX

Purpose

Selects the index to be used with the current file or returns the name of the current index for the current file.

Syntax

INDEX *index* / *strexpr*

Comments

Superbase automatically selects the first index as default when you open a file or when you make an open file the current file. With the INDEX command, you can select another of the file's indexes.

index is the name of a field in the current file; *strexpr* must evaluate to the name of a field in the current file. An index for this field must already exist.

Because DML parses entire lines before executing them, you cannot place an INDEX statement – or any other statement that refers to a field – on the same line as the OPEN FILE statement. When the DML interpreter reaches the INDEX statement, the file will not yet be open, and the field name, therefore, will not be recognized. As a result, a line like:

```
OPEN FILE "aaa":INDEX anum
```

produces an error message.

INDEX also acts as a system variable and holds the name of the current index in the current file.

Examples

- 1 OPEN FILE "aaa"
INDEX anum
- 2 a\$ = "Lastname"
INDEX a\$
- 3 IF INDEX = "Country" THEN . . .

Notes

Example 3 shows the use of INDEX as a system variable.

See Also

? INDEX, CREATE INDEX, FILE, OPEN INDEX

INPUT

Purpose

Reads characters or a line from a text file on disk into a variable or a field.

Syntax

INPUT [*&nexpr*[,] / LINE] *var*/*field*

Comments

This statement inputs data from a text file on disk. It assumes that an input channel has been opened – for example, by OPEN FOR INPUT.

With the LINE option, INPUT reads data from the input channel until it finds a line feed character (ASCII 10) and inputs the data into *var* or *field*. If the line feed character has been used as the record separator, INPUT LINE reads a record at a time.

var can be a string or numeric variable and *field* can be a text, numeric or date field. However, in each case the type of variable or field used should match the type of data expected. Thus if you attempt to input alphabetic characters into a numeric variable they will be valued at zero, while inputting them into a date field will produce an invalid date error.

&nexpr specifies the number of characters that the command takes from the text file. The comma after *nexpr* is entirely optional.

When *nexpr* is zero or when *&nexpr* is not used, the command reads characters from the text file until it finds a comma (ASCII 44) or a line feed character (ASCII 10). This option allows INPUT to be used instead of IMPORT to read in data from an exported file. To detect the end of a text file, use EOF("*").

Examples

```
1  INPUT LINE a$
2  INPUT &6a$ (or INPUT &6,a$)
3  INPUT a%
4  FILE "aaa": EXPORT TO "aaa.exp"
   OPEN "aaa.exp" FOR INPUT
   FOR i% = 1 TO RECCOUNT("aaa")
   BLANK
   INPUT field1
   INPUT field2
   .....
   INPUT lastfield
```

STORE
NEXT i%: CLOSE INPUT

Example 4 demonstrates how INPUT can be used as an alternative to IMPORT.

See Also

?, CLOSE, IMPORT, OPEN FOR

INSTR

Purpose

Returns the starting character position of a substring within a string, or returns 0 if the substring is not contained within the string.

Syntax

INSTR ([*nexpr* ,] *strexpr* , *substrexpr*)

Comments

This function returns the position in *strexpr* of the first occurrence of *substrexpr*. If *nexpr* is used, INSTR searches for the first occurrence of *substrexpr* from position *nexpr* onwards. If the substring is not found, the value returned is zero.

nexpr must be positive and less than the length of *strexpr*. INSTR is case sensitive, i.e., "abc" and "ABC" are different.

Examples

```
1  x%% = INSTR(x$, y$)
2  x%% = INSTR(textfield, "Mr")
3  x%% = INSTR(x$, "Mr"): y%% = INSTR(x%% + 1, x$, "Mr")
4  ? LEFT$(textfield, INSTR(textfield, " ") - 1)
5  x$ = LCASE$(textfield): x%% = INSTR(x$, "abc")
   IF x%% > 0 THEN ? LEFT$(textfield, x%% - 1)
```

Notes

The second example simply locates the title 'Mr'. Example 3 locates any 'Mr and Mrs' or 'Mr & Mrs'. Example 4 displays the first word in *textfield*. Example 5 finds any occurrence of the sequence 'abc' in *textfield* regardless of what case the letters are in.

INT

Purpose

Removes the part of a number to the right of the decimal point, turning it into a whole number.

Syntax

INT (*nexpr*)

Comments

INT does not round a decimal number up or down to the nearest whole number, but simply strips off the decimal part. Thus INT(123.00001) and INT(123.99999) give the same result – 123.0.

Examples

- 1 numfieldc = INT(numfielda)
- 2 numfieldc = INT(numfielda * (1 + numfieldb) / 100)
- 3 x%% = INT(VAL(RIGHT\$(textfield, 2)) / 3.33)
- 4 numfieldc = INT(numfielda + 0.5)
- 5 numfieldc = INT(numfielda + 0.5 * SGN(numfielda))

Notes

Example 1 sets *numfieldc* equal to the integer part of *numfielda*, while Example 4 sets *numfieldc* to the nearest integer number to *numfielda* (for positive values only), and Example 5 performs similar rounding on any value..

See Also

FIX

KEY

Purpose

Displays the current set of function key definitions, or defines a new set.

Syntax

KEY *keynum* [,*string*]

Comments

On the keyboard the function keys are F1 to F10. Each of these can be used alone or with SHIFT, CTRL or SHIFT+CTRL, giving 40 in all. Each key has a *keynum*. The *keynums* associated with the keys are:

(F1 - F10)	1-10
SHIFT+(F1 - F10)	11-20
CTRL+(F1 - F10)	21-30
SHIFT+CTRL+(F1 - F10)	31-40

string can be any set of Superbase commands which can be entered on one line (provided that they do not access a field on the same line as an OPEN FILE). The command line in *string* is assigned to the key specified by *keynum*.

Using KEY without a following command string clears the key associated with *keynum*. If *keynum* and *string* are not used, KEY displays the current set of function key definitions.

Examples

- 1 KEY 11
- 2 KEY 22, "OPEN FILE ~aaa~:BLANK:ENTER:STORE:
? ~NOW~; RECCOUNT(~aaa~); ~Records~"
- 3 KEY
- 4 KEY 3, "Tel. (0428) 725400 [(04203) 5601 evenings]"

Example 1 clears key SHIFT+F1. Example 2 sets CTRL+F2 to enter a record into a file and report how many records there now are. Note the use of the tilde character to insert quotation marks. Since the entire string must be enclosed in quotation marks, you cannot use quotation marks within the string.

Notes

Preface the key string with the '!' exclamation point to generate a function key that will always be executed rather than output as text.

See Volume 1, Chapter 29 Function Keys for further information.

See Also

LOAD KEY, SAVE KEY

LABELS

Purpose

Prints records as labels.

Syntax

LABELS [FILE *sbfname*] [WHERE *conditions* / ASK] [USING *labelparams*]

Comments

This command is the program equivalent of selecting the Labels command from the Process menu. It allows you to print out records as labels and to define their format.

WHERE *conditions* limits the records for which labels are printed and acts as a filter.

USING allows you to specify the shape and content of the labels to be printed. *labelparams* is a series of parameters separated by commas, relating to the label definition requester.

line 1 fields/line
line 2 fields/line
line 3 fields/line
line 4 fields/line
line 5 fields/line
line 6 fields/line
line 7 fields/line
line 8 fields/line
line 9 fields/line
line 10 fields/line
First label margin
Label text width
Second label margin
First line next label
Copies per label
Labels per line
Top label margin
Labels before FF

Note: All 18 parameters must be used.

If USING is not specified, LABELS takes the default parameters as shown in the label definition requester (see Volume 1, Chapter 26 Label Printing).

Examples

- 1 LABELS USING 1,0,2,1,1,1,1,1,0,0,1,35,40,12,1,2,0,0
- 2 LABELS FILE "Address" WHERE Lastname LIKE "[a-c]**"

Notes

The LOAD LABELS command may be used to set the default labels format for use with LABELS without a USING clause.

LCASE\$

Purpose

Converts a text string to lowercase.

Syntax

LCASE\$ (*strexpr*)

Comments

This function changes upper case letters to lower case; no other characters, including those already in lowercase, are affected.

The complementary function to LCASE\$ is UCASE\$.

Examples

- 1 textfieldc = LCASE\$(textfielda)
- 2 x\$ = LCASE\$(y\$)
- 3 x\$ = LCASE\$("ABCDEF")
- 4 ? LCASE\$(x\$)

See Also

FCASE\$, UCASE\$

LEFT\$

Purpose

Extracts one or more characters from a string, starting at the left of the string, or transfers spaces to the end of the string.

Syntax

LEFT\$ (*strexpr* [,*nexpr*])

Comments

LEFT\$ returns the leftmost *nexpr* characters of the string *strexpr*. Thus, if *strexpr* is **DICTIONARY** and *nexpr* is 7:

```
LEFT$("DICTIONARY", 7)
```

returns **DICTION**.

If *nexpr* is omitted, any leading spaces in *strexpr* will be transferred to the end of the string.

Examples

- 1 textfieldc = LEFT\$(textfielda, 10)
- 2 textfieldc = UCASE\$(LEFT\$(textfielda, 1)) + MID\$(testfielda, 2)
- 3 LEFT\$(textfielda, 1) LIKE [a-c]
- 4 x\$ = LEFT\$("ABCD", 2)
- 5 x\$ = LEFT\$(x\$, n%%)
- 6 ? LEFT\$(x\$, 10)
- 7 strip: IF LEFT\$(x\$, 1) = " " THEN x\$ = MID\$(x\$, 2): GOTO strip

Notes

Example 7 is a one line program to strip leading spaces from x\$ (see **LTRIM\$**).

See Also

LEN, MID\$, RIGHT\$

LEN

Purpose

Returns the number of characters in a text string or text field.

Syntax

LEN (*strexpr*)

Comments

LEN counts the number of characters in a string, including spaces and non-printing characters.

Examples

```
1  numfieldc = LEN(textfielda)
2  textfieldc = RIGHT$(textfielda, LEN(testfieldb))
3  IF LEN(textfielda) > 25 AND LEN(textfielda) < 50 THEN ...
4  x%% = LEN(x$)
5  x%% = LEN(MID$(textfield, 3))
6  ? LEN(x$)
```

See Also

LEFT\$, MID\$, RIGHT\$

LET

Purpose

Assigns a value to a variable.

Syntax

[LET] *var / field* = *expr* / [*expr1* ? *expra* : *exprb*]

Comments

The keyword LET is optional and is usually omitted. In other words, to assign a value to a variable or a field, you only need to use the equal sign.

The LET option has been included in DML to maintain compatibility with most versions of BASIC. DML also provides a more unusual facility when you use the syntax:

var / field = *expr1* ? *expra* : *exprb*

This option provides a short way of assigning different values to a variable or field, depending on whether an expression is true or not; that is, it makes assignment conditional on the truth or falsity of a specified expression. If *expr1* is true, *expra* is assigned to the variable or field; if it is false, *exprb* is assigned. It is equivalent to

```
IF expr1
THEN varfield = expra
ELSE varfield = exprb
```

Used in this way, the question mark character is referred to as a 'ternary operator' to reflect the fact that three operands are required at the right of the equal sign. In fact, you can chain ternary operators together to create a statement which contains multiple conditions and assigns one of multiple values.

Within a program, however, you will generally find it easier to use the IF THEN ELSE or SELECT CASE statements. The main application for the ternary operator is in a Superbase file definition. Here it has an important advantage over the IF THEN ELSE statement: it can be entered as a calculation formula for a field in a file definition.

For a fuller discussion of the ternary operator and its applications, see Volume 1, Chapter 8 Derived Values.

Examples

- 1 item\$ = "Sprocket"
- 2 Textfielda = "London"
- 3 b#% = 3.25
- 4 numfieldb = INT(277/62)

The following examples assume that x\$="ABC" and x#%=4.5:

- 5 y\$ = (x\$ = "ABC") ? "yes" : "no"

Assigns 'yes' to y\$.

- 6 y\$ = (x\$ = "aaa") ? x\$: ""

Assigns '' to y\$.

- 7 y\$ = (x#% > 3.5) ? "yes" : "no"

Assigns 'yes' to y\$.

- 8 y\$ = (EOF ("")) ? "end of file" : "more to read"

LIST

Purpose

Lists a text file to the screen.

Syntax

LIST *filename*

Comments

LIST is the program equivalent of the LIST command in the Utilities menu. It displays a text file on the screen.

Do not confuse LIST with ? LIST, which displays a program listing. LIST only works with text files. Note also that unlike many other DML commands, LIST requires the file name itself plus its extension name.

Examples

- 1 LIST "aaa.exp"
- 2 LIST "Address.sbd"

See Also

? DIRECTORY, ? LIST, OPEN FOR INPUT

LOAD

Purpose

Loads any of the following types of file into memory: program, text, labels format, function key definitions, Query, Update or Superbase parameters.

Syntax

LOAD [TEXT]/[KEY]/[QUERY]/[UPDATE]/[LABELS]/[SET] [*filename*]
[,APPEND]

Comments

If none of the options TEXT, KEY, LABELS, QUERY, UPDATE or SET is used, Superbase assumes that *filename* refers to an SBP file and attempts to load a PROGRAM file.

The APPEND option can be used with program files and text files to append a file on disk to the file in memory.

Note that LOAD cannot be used to load a Superbase data file (a file with an SBF extension). For this you need to use the OPEN FILE command.

LOAD SET causes the Superbase parameters file S:SUPERBASE.INI to be re-loaded into memory from disk. *filename* should not be specified with SET.

Examples

- 1 LOAD "Program1"
- 2 LOAD TEXT "Banklet"
- 3 LOAD QUERY "Deptran"
- 4 LOAD KEY "Funkey1"
- 5 LOAD TEXT "Document2", APPEND
- 6 LOAD LABELS "labels1"
LABELS

Notes

Example 6 loads the LABELS1.SBB file and outputs using this as the default format.

See Also

?, CHAIN, SAVE

LOCATE

Purpose

Sets the position at which the next output appears on the current output device (generally, the screen or the printer).

Syntax

LOCATE *column* [,*row*]

Comments

When you are displaying something on the screen, LOCATE allows you to specify where it appears.

column and *row* must be numeric expressions. *row* is optional and if it is not included, LOCATE moves the cursor to the specified column position on the current line.

You cannot use LOCATE to move the cursor backwards or up the screen: it will not move the cursor to a position on the same line which is to the left of the current position; and it will not move the cursor to a line above the current position. However, you can bypass this restriction if you place the HOME statement in front of LOCATE – see HOME.

Examples

- 1 LOCATE 12,6: ? "Hello"
- 2 c%% = 12: r%% = 1: LOCATE c%%,r%%
- 3 LOCATE 12: ? "Hello"

Notes

The first example displays the word 'hello' at the 12th column on the sixth row. Example 3 displays 'hello' at the 12th column of whatever line the cursor happens to be on.

See Also

? @, ASK, CLS, COL, HOME, ROW

LOG

Purpose

Returns the natural logarithm (log to the base 'e') of a number.

Syntax

LOG (*expr*)

Comments

The value of *expr* must be positive. Negative numbers or zero produce the error message 'Invalid numeric parameter.'

Examples

- 1 numfieldc = LOG(numfielda)
- 2 numfieldc = LOG(datefielda - datefieldb)
- 3 IF numfieldc > LOG(numfielda * numfieldb) THEN ...
- 4 x#% = LOG(y#%)

5 $x\#\% = \text{LOG}(y\#\% * \text{numfielda} * (\text{datefielda} - \text{datefieldb}))$

See Also

EXP

LOOKUP

Purpose

Detects whether an expression occurs in a file in a specified field.

Syntax

LOOKUP (*expr* , *field* , *file*)

Comments

This function checks whether a given field in a file contains the expression in *expr*. That is, it answers the question: does *expr* exist in *field.file*? If it finds the expression, LOOKUP returns the value - 1 (true); otherwise it returns the value 0 (false).

expr and *field.file* must be of the same type (i.e. text or numeric). *field.file* must be the name of an indexed field in an open file.

LOOKUP plays a major role in cross-file validation and calculation. The file specified with LOOKUP can be any open file; so you can use this function to perform a relational lookup, where it checks whether the contents of a field in one file match the contents of a field in another file.

If LOOKUP is successful – if it finds the expression you have specified – the record containing the expression becomes the current record, even though the file may not be the current file and is not displayed on screen. At the same time, the FOUND function is set to -1 to reflect the fact that the search has been successful.

The LOOKUP function is case sensitive; for example:

```
LOOKUP("fred", Firstname. Address)
```

will not find an occurrence of 'Fred.'

For more information on LOOKUP, see Volume 1, Chapter 18 Linking Files.

Examples

```
1 OPEN FILE "aaa": OPEN FILE "bbb"
  SELECT FIRST
  WHILE NOT EOF("bbb")
```

```
IF LOOKUP (field1.bbb, field1.aaa) THEN GOSUB Process_module
WEND
? "Process completed": END
Process_module:
.....
RETURN
```

This example demonstrates a small program to process only those records in file 'bbb' that have a matching record in file 'aaa.' File 'bbb' could be an invoice file joined to a customer file 'aaa' by a customer code (field1 in 'bbb' and 'aaa').

See Also

EXISTS

LTRIM\$

Purpose

Trims leading spaces from a text expression or a text field.

Syntax

LTRIM\$ (*strexpr*)

Comments

LTRIM\$ returns a string consisting of the original string specified by *strexpr* with any leading spaces removed.

Examples

```
1  textfieldc = LTRIM$(textfielda)
2  x$ = LTRIM$(textfieldc.filea)
3  ? LEN(x$); LEN(LTRIM$(x$))
```

See Also

TRIM\$

MACRO

Purpose

This command is included for compatibility only. It may be used in DML programs written for other computers, but on the Amiga it has no effect.

MAKE

Purpose

Stores the file definition for a file after it has been defined by CREATE and ADD.

Syntax

MAKE *sbfname*

Comments

This command is used as the last step in the process of creating a new file. After the file has been defined by CREATE and ADD, MAKE writes the new file definition to disk together with any indexes that have been created.

Note that a file definition is not regarded as valid until the MAKE command has been executed. Before then, any error will have the effect of removing the file definition.

Examples

```
1  CREATE "Address"
   ADD "Firstname;TXT REQ;20;1;2"
   ADD "Lastname;TXT REQ IXD;20;1;33"
   ADD "Street;TXT;25;3;2"
   .....
   (other field definitions)
   .....
                                     MAKE "Address"
```

See Also

ADD, CREATE, CREATE INDEX

MAX

Purpose

Returns the maximum value from a range of values.

Syntax

MAX (*fieldname* / *numarray*)

Comments

fieldname must be the name of a numeric field which is repeated in a report or in a transaction form. *numarray* must be a numeric array which has been defined in a program or as calculation variable in a transaction line. The parentheses are optional. See COUNT.

Examples

- 1 ? MAX range#%
- 2 a%% = MAX Score.Results

See Also

COUNT, DIM, GROUP, MEAN, MIN, REPORT, SD, SUM, VAR

MEAN

Purpose

Returns the average from a range of numeric values.

Syntax

MEAN (*fieldname* / *numarray*)

Comments

fieldname must be the name of a numeric field which is repeated in a report or a transaction form. *numarray* must be a numeric array which has been defined in a program or as a calculation variable in a transaction line. The parentheses are optional.

Examples

- 1 a#% = MEAN age

```
2  DIM m#%(10)
   FOR n%% = 0 to 10
     m#%(n%%) = n%%
   NEXT
   ? MEAN m%
```

See Also

COUNT, DIM, GROUP, MAX, MIN, REPORT, SD, SUM, VAR

MENU

Purpose

Defines a line in a user-defined menu.

Syntax

MENU *menu-id*, *item*, *state* [,*strexpr*]

Comments

Superbase allows you to define up to 10 menus each of which can have up to 12 items. With the MENU command you supply the text for a single item and specify whether it will appear on the menu as enabled, disabled (ghosted) or with a check mark against it. Having defined a menu with a series of MENU commands – one for each menu item and one for each menu title – you can then use the MENU ON command to turn the menu (or menus) on. You also use MENU ON to specify a numeric variable which will return a value showing which item, if any, has been selected.

menu-id must be a numeric expression with a value in the range 1 to 10 identifying one of the ten possible menus. Menus are numbered from the left as they appear on the screen. To define a line in the leftmost menu (displayed in the same position as the Superbase Project menu), you would enter a value of one for *menu-id*.

item must be a numeric expression with a value in the range 0 to 12, giving the number of the menu item. Item 0 is the menu heading, the text that appears on the menu bar. Item 1 defines the width of the menu in terms of the number of characters. When you enter the text string parameter for this item you may need to allow for the width of other items in the menu by adding spaces to the end of the string.

state can take a value of 0, 1 or 2. 0 disables the item so that it appears on the menu as a ghosted option. 1 enables it, 2 places a check mark against it.

strexpr supplies the text for the item. For example, if you wished to define a line in the first menu which contained the option Deposits, you could enter:

```
MENU 1, 3, 1, "Deposits"
```

This would make Deposits the third item in the menu. To disable the Deposits option, you would enter:

```
MENU 1, 3, 0
```

Note that you do not need to specify the text a second time.

If you wish to use checkmarks in your menus, remember to leave a space for the checkmark character when you define the menu text strings – the first character in each string should be a space.

Each string may also contain an ampersand ('&'). The ampersand is not displayed, but indicates to Superbase that the next character is to be used as a 'keyboard equivalent' for this menu item. Pressing the specified key while holding down the right Amiga key is equivalent to selecting this menu command with the mouse.

When Superbase displays a menu, any keyboard equivalents are shown at the right hand side of the menu, following an Amiga key symbol. You would usually make the keyboard equivalent key the same as the initial letter of the text string, but you may need to choose a different character to avoid ambiguity.

MENU on its own clears all the menu settings (as do ERASE and CLEAR).

Examples

```
1  MENU 2, 0, 1, "Transactions  "  
    MENU 2, 1, 1, "&Deposits"  
    MENU 2, 2, 1, "&Withdrawal"  
    MENU 2, 3, 1, "D&irect debit"  
    MENU 2, 4, 1, "&Standing orders"  
    MENU 2, 5, 1, "&Credit card"  
    MENU ON a%%, b%%
```

Notes

The example defines the second menu with five options (items), all of them enabled. (The heading for the menu is Transactions.) MENU ON then turns the menu on. When the user selects an item, Superbase places its menu-id and item numbers in the variables a%% and b%%.

See Also

MENU ON, SET MENU

MENU CLEAR

Purpose

Turns off all user-defined menus and clears their definitions from memory. Also clears the menu bar display.

Syntax

MENU CLEAR

Comments

If you want to define a new set of menus, you can use this command to clear any menus which have been defined previously. You may also use it when your menus are no longer required, in order to make the memory space they occupy available for other purposes.

See Also

MENU OFF, SET MENU

MENU ON/OFF

Purpose

Turns user-defined menus on and specifies the variables which return the result of a menu selection.

Syntax

MENU ON *nvar1*, *nvar2*

MENU OFF

Comments

MENU ON turns on any menus which have been defined with the MENU command and sets *nvar1* and *nvar2* to zero.

When the user selects an item, Superbase places the menu-id number in *nvar1* and places the selected item number in *nvar2*. It also turns all the user-defined menus off.

For example, if the second item in the third user-defined menu has been selected, the first numeric variable specified with MENU ON, will contain the value 3 and the second numeric variable will contain the value 2.

After the MENU ON command has been executed, any menus that have been defined remain active until an item has been selected. The MENU ON command should then be executed again in order to reset the menu variables to 0.

You may sometimes want to turn the menus off without waiting for an item to be selected. You can do this with the command MENU OFF.

Examples

```
1  menuloop:
    MENU ON a%%, b%%
    WAIT MENU
    ON a%% GOSUB sub1, sub2, sub3, sub4, sub5
    GOTO menuloop
```

Notes

This example presumes that five menus have been defined and that **sub1** to **sub5** are subroutines which handle item selection for each menu.

See Also

SET MENU, WAIT

MERGE

Purpose

Loads a text file and performs a mail merge.

Syntax

MERGE [TEXT *filename*] [WHERE *conditions*]

Comments

This command merges the data in an SBF file (a database file) with a form letter in the Text Editor and outputs the results to the printer. It takes data from the current open file and – if WHERE is not included – prints one letter for each record in the file. You can use WHERE to set up a filter restricting the merge operation to only those records which match the conditions specified. The TEXT option lets you specify a text file on disk; Superbase will then load the file into the Text Editor before starting the merge operation.

Although Merge is the program equivalent of the Mail Merge option on the Process menu, it does not allow you to preview letters on screen before printing them. To do this, use ? TEXT with the MERGE parameter.

Examples

```
1 OPEN FILE "Address"  
MERGE TEXT "Mailshot1" WHERE Country LIKE "USA"
```

See Also

? TEXT

MID\$

Purpose

Returns one or more characters from within a text string or text field.

Syntax

MID\$ (*strexpr* , *nextpr1* [,*nextpr2*])

Comments

MID\$ is more flexible than LEFT\$ and RIGHT\$ as it can extract characters from any point in a string. *strexpr* holds the string, and *nextpr1* gives the starting point in the string. *nextpr2* specifies the length of the substring to be extracted; if *nextpr2* is not given, MID\$ takes all the characters from the starting point to the end.

Examples

```
1 textfieldc = MID$(textfielda, 10, 10)  
2 textfieldc = LCASE$(MID$(textfielda, 8))  
3 IF MID$(textfielda, 12, 1) LIKE "[a-c]" THEN ...  
4 x$ = MID$(textfieldc, 19, 2)  
5 x$ = MID$(x$, 4)  
6 ? MID$(x$, 4, 2)  
7 ASK; a$  
  l%% = LEN(a$)  
  b$ = ""  
  FOR n%% = l%% TO 1 STEP -1  
    b$ = b$ + MID$(a$, n%%, 1)  
  NEXT  
  ? b$
```

Notes

Example 7 inputs a word into A\$ and turns it back to front.

See Also

LEFT\$, LEN, RIGHT\$

MIN**Purpose**

Returns the minimum value from a range of numeric values.

Syntax

MIN (*fieldname* / *numarray*)

Comments

fieldname must be the name of a numeric field which is repeated in a report or in a transaction form. *numarray* must be a numeric array which has been defined in a program or as calculation variable in a transaction line. The parentheses are optional. See COUNT.

Examples

- 1 IF MIN range#% < 15 THEN ...
- 2 min%% = MIN Score.Results

See Also

COUNT, DIM, GROUP, MAX, MEAN, REPORT, SD, SUM, VAR

MINS**Purpose**

Extracts the number of minutes (as would appear in a 'hh:mm:ss' display) from a numeric value which holds the time in thousandths of a second.

Syntax

MINS(*nexpr*)

Comments

Usually, *nexpr* will be a timefield or the result of a TIMEVAL calculation.

Examples

```
1 mnts%% = MINS(timefield)
```

```
2 ? MINS (NOW - start&%) ;" minutes have elapsed"
```

See Also

HOURS, SECS, THOUSECS

MOD

Purpose

Gives the remainder of a numeric expression after it has been divided.

Syntax

nexpr1 MOD *nexpr2*

Comments

nexpr1 is the number to be divided, *nexpr2* is the number that divides into it (the divisor). MOD returns the remainder when *nexpr1* has been divided by *nexpr2*. For example:

```
14 MOD 3
```

gives 2 as a result. It is equivalent to:

```
14 - INT (14/3) * 3
```

Examples

```
1 ? (2.53 * 100) MOD 100
```

Notes

The example line strips off the integer part of a number and displays the first two figures after the decimal place.

MODIFY

Purpose

Modifies a field definition.

Syntax

MODIFY *field* [,] [field definition string] [*formula*] [*formula*]

Comments

MODIFY is the program equivalent of the Modify File command on the Project menu. It allows you to alter a field's parameters; for example, the field name or its length. The field definition and formula strings take the same form as they do with the ADD command. This command cannot be used to modify an indexed field or to add an index to a field.

Examples

```
1  MODIFY Forename "Firstname;TXT REQ;15 U;1,12"
```

See Also

ADD, MAKE, SAVE FILE

MONTH

Purpose

Takes a julian date number and returns a numeric value for the month of the year.

Syntax

MONTH (*nexpr*)

Comments

This function is valid for dates from January 1, 1 A.D. to December 31, 9999 only. Consequently *nexpr* is valid only in the range 1 to 3652048. If *nexpr* is 0, MONTH returns 0. If *nexpr* is negative or is greater than 3653048, the result is unpredictable.

Examples

```
1  numfieldc = MONTH (datefielda)
2  numfieldc = MONTH (datefielda + 90)
3  numfieldc = MONTH (TODAY)
```

4 x%% = MONTH(datefielda + VAL(textfielda))

5 x%% = MONTH(DAYS("11 Jan 85"))

Notes

Example 3 provides a calculation to insert the month number of the system date into a numeric field.

See Also

DAY, DAYS, DAY\$, DATE\$, MONTH\$, YEAR

MONTH\$

Purpose

Takes a julian date number and returns the month of the year as a text string.

Syntax

MONTH\$ (*nexpr*)

Comments

The same limitations on which julian dates are acceptable apply to this function as they do to other date functions.

The format of the text string is the full month name regardless of what current date format is – i.e., January, not Jan.

Examples

1 textfieldc = MONTH\$(datefielda)

2 textfieldc = MONTH\$(datefielda + 90)

3 textfieldc = MONTH\$(TODAY)

4 x\$ = MONTH\$(datefielda + VAL(textfielda))

5 x\$ = MONTH\$(DAYS("11 Jan 85"))

6 ? MONTH\$(datefielda + 30)

See Also

DATE\$, DAY, DAYS, DAY\$, MONTH, YEAR

MOUSE

Purpose

Reads the position of the mouse pointer and responds to mouse button input.

Syntax

MOUSE *nvar1*, *nvar2*, *nvar3*

Comments

You may use this command to detect the position of the mouse pointer on screen and to check whether the mouse button has been clicked or double clicked.

nvar1 returns the state of the mouse button. The possibilities are:

- 0 no button has been pressed since the MOUSE command
- 1 button has been pressed and released
- 2 button has been pressed and released twice within the doubleclick period understood by Superbase
- 3 button is held down now and the mouse is moving.

nvar2 returns the X coordinate position of the mouse pointer in pixels, relative to the top left-hand corner of the window. *nvar3* returns the Y coordinate position of the mouse pointer in pixels, relative to the top left-hand corner of the window.

The first call of the MOUSE command informs Superbase of the variables used to track the mouse position. Subsequent use of the command will return the current position and state in those specified variables.

The state variable is continuously updated by Superbase, but the position variables only reflect the mouse position when the button last changed state or the MOUSE command was executed; i.e., if *nvar1* has the value 1 or 2, *nvar2* and *nvar3* return the current position of the mouse pointer when the button is released.

The first time you execute MOUSE, you use it as a dummy to set up the variables and start Superbase tracking MOUSE clicks.

Whenever you read *nvar1* it returns the last state of the MOUSE button. Once read, *nvar1* retains its value until another MOUSE event occurs, so you may want to reset it to 0 with the command '*nvar1*=0' or by executing the MOUSE command again.

Note: The MOUSE command cannot detect the operation of the right or centre buttons.

Examples

```
1  MOUSE ON
   MOUSE n%%,x%%,y%%
   WHILE n%% = 0
   MOUSE n%%,x%%,y%%
```

```
IF x%% > 300 THEN PANEL OFF ELSE PANEL ON
WEND
PANEL ON

2  MOUSE ON
   MOUSE n%%,x%%,y%%
   WAIT MOUSE
   IF x%% > 300 THEN PANEL OFF ELSE PANEL ON
```

Notes

If you want to monitor the current position of the mouse pointer you must execute the MOUSE command in a WHILE WEND loop. In the first example, the program simply turns the browsing controls panel on and off according to the pointer position. When the pointer is moved to the right of the screen, the panel is removed; and it is restored when the pointer is moved to the left. The second example works in the same way except that the program waits for a mouse click in the database window.

See Also

WAIT MOUSE

MOUSE ON/OFF

Purpose

Changes the mouse pointer from a watch shape to an arrow shape.

Syntax

MOUSE ON / OFF

Comments

Normally, when a program is running, the mouse pointer is shown as a representation of a watch. With this command, you can change the shape of the mouse pointer from the watch to an arrow. The arrow is easier to use when selecting objects on screen.

ON produces the arrow; OFF produces the watch.

NEW

Purpose

Clears the program area or text area.

Syntax

NEW [TEXT/QUERY/UPDATE]

Comments

On its own, NEW erases any program that is currently in the computer's memory. When followed by TEXT, it clears the current text editor area of memory. Following it by QUERY or UPDATE, clears their respective requesters.

Unlike the DML menu New command this command does not put you into the program editor.

See Also

CHAIN, LOAD, SAVE

NEWLINE

Purpose

Sends a new line character (or characters) to an output device.

Syntax

NEWLINE [*nexpr*]

Comments

This command prints a new line at the current output device; i.e. with the screen display, it takes the cursor onto the start of the next line. *nexpr* can be used to specify more than one new line. If *nexpr* is not an integer, only the integer part will be taken.

Examples

```
1  NEWLINE 2
2  FOR i%% = 1 TO 20
   ? i%%
   if i%% MOD 5 THEN NEWLINE i%%/5
   NEXT i%%
```

Notes

Example 2 outputs the numbers 1 to 5 with single line spacing, 6 to 10 with double spacing, and so on up to 20.

See Also

EJECT, LOCATE, PCOL, PROW

NOW

Purpose

Allows the current time to be displayed, printed or used in calculations.

Syntax

NOW

Comments

NOW is a numeric system variable which contains the current time, expressed in thousandths of a second. Its value may be assigned to a numeric variable or field as shown in Example 3.

If you have a real-time clock, or you have set the system time, the value of NOW is the number of thousandths of a second since midnight. Otherwise the value of NOW is the number of thousandths of a second since you last started up or reset your computer.

If you display the value of NOW, as shown in Example 1, Superbase automatically translates the number into minutes and seconds, which it displays in the current time format – just as though you had typed:

? TIME\$(NOW)

To set the time, use the Date format requester or the SET NOW command.

Examples

- 1 ? NOW
- 2 ? MINS(NOW)
- 3 tvar% = NOW
? NOW; &8.0 tvar%, TIME\$(tvar%)
- 4 tt% = NOW
FOR n%% = 1 TO 1000

NEXT

? "Elapsed time is: ";TIME\$(NOW - tt&%, "hh:mm:ss.s")

Notes

Example 4 demonstrates the use of NOW and a numeric variable (tt%) to time an action or an event – in this case, the time taken for Superbase to execute an empty FOR NEXT loop.

See Also

TIME\$, TODAY

NUMBASE

Purpose

Sets the numeric format in which numbers are displayed.

Syntax

NUMBASE *string*

Comments

NUMBASE is the program equivalent of the Number Format option in the Set menu. *string* must be one of Superbase's valid numeric formats, as listed in Volume 1, Chapter 3. For example, "z99999.00" or "z(+ \$,000000.00)". See also STR\$ and the & output format parameter.

Examples

1 NUMBASE "z99999."

Integer only format.

2 NUMBASE "+*****.00"

Numbers displayed with a sign and leading check protection characters.

See Also

DATEBASE, STR\$

ON ERROR

Purpose

Tells DML to branch to another part of the program when an error occurs.

Syntax

ON ERROR [[GOTO] *label*]

Comments

Normally, DML halts program execution and displays an error message when it detects an error. ON ERROR enables error trapping, and prevents the program from halting. Once an error has been detected, it causes the program to jump to the error handling routine specified with *label*.

You can use ERRNO in your error handling routine to check which error has occurred, and take appropriate action. In many cases, you will want the program to resume execution after detecting an error. You can do this with the RESUME statement.

To disable error trapping, use ON ERROR without a following label.

Examples

```
1  ON ERROR GOTO check
    .....
    .....
    check: IF ERRNO <> 11 THEN
        REM Note - errors other than 11 now go unreported
        RESUME
    ELSE
        ? "Are you sure you want to exit from this program?"
        ? "Press Y to exit, any another key to resume"
        WAIT a$
        IF a$ = "Y" OR a$ = "y" THEN END ELSE RESUME
    ENDIF
```

Notes

In this example, ON ERROR is used to check whether the Stop button has been clicked or CTRL+C has been pressed. Both these events generate error number 11, so the error handling routine (which starts at label 'check') first tests for this error number. If it finds that another error event has occurred, program execution is resumed at the line which caused the error. The error handling routine then asks if the user wishes to exit or not.

Depending on the answer it receives, it either resumes execution at the line which caused the error (the line being executed when the user clicked Stop or CTRL+C) or terminates the program.

See Also

ERR\$, ERRNO, RESUME

ON GOSUB

Purpose

Calls one of a number of subroutines from a list of subroutines.

Syntax

ON *expr* GOSUB *label1* [, *label2*, *label3*, ...]

Comments

This statement transfers program control to one of the subroutines given in the list. The value of *expr* determines which subroutine the program jumps to. If *expr* has valued at 1 the program branches to the subroutine at the first label, if *expr* has a value of 2, it branches to the subroutine at the second label, and so on.

Once the program has branched to a subroutine, it executes each statement in turn until it meets a RETURN statement. Then it jumps back to the line following the ON GOSUB statement.

Any label can be repeated.

If *expr* is 0 or greater than the number of supplied labels, program control drops to the next statement after the ON GOSUB statement.

Examples

```
1  ON x%% GOSUB lab1,lab2,lab3
2  ON x%% GOSUB lab1,lab2,lab1,lab2,lab1
```

See Also

GOSUB, RETURN, SELECT CASE

ON GOTO

Purpose

Branches to one of a list of labels.

Syntax

ON *nexpr* GOTO *label* [*label*,....]

Comments

This command transfers program control to one of the program lines given in the list. The value of *nexpr* determines which label the program jumps to. If *nexpr* has a value of 1 the program branches to the first label, if *nexpr* has a value of 2, it branches to the second label, and so on. For a general description of GOTO refer to GOTO itself.

nexpr should be a positive integer. If it is not an integer, the whole number part will be taken.

Any label can be repeated.

If *nexpr* is 0 or greater than the number of supplied labels, program control drops to the next statement after ON GOTO.

Examples

```
1  ON x%% GOTO lab1,lab2,lab3
2  ON x%% GOTO lab1,lab2,lab1,lab2,lab1
   ? "Reaches here only when x%% is 0 or greater than 5"
```

See Also

GOTO

OPEN

Purpose

Opens a text file on disk for input/output.

Syntax

OPEN *filename* FOR [INPUT / OUTPUT / APPEND]

Comments

OPEN *file* FOR OUTPUT creates an ASCII file on disk in the same way as OUTPUT TO *file*. The difference is that OUTPUT TO includes page formatting information in its output (as required by the current page format), and OPEN *file* FOR OUTPUT writes a sequential file containing the specified information and nothing else.

There is only one channel for INPUT, and one for OUTPUT, so you cannot have two output channels, or two input channels. However, you can have one of each open at the same time.

APPEND is an output channel and specifies that *file* is to be appended to. You must not try to specify OPEN "aaa" FOR OUTPUT APPEND.

If using OUTPUT, *file* is overwritten without warning.

If using APPEND, *file* need not exist.

If using INPUT, *file* must exist.

After an OPEN *file* FOR OUTPUT/APPEND command, every ? command sends its output to *file* until the CLOSE OUTPUT command is executed.

Examples

```
1  OPEN "aaa" FOR OUTPUT
2  OPEN "aaa" FOR APPEND
3  OPEN "bbb" FOR INPUT: OPEN "aaa" FOR APPEND
   labl1: INPUT LINE a$: ? a$
       IF NOT EOF ("") THEN GOTO labl1
       CLOSE INPUT: CLOSE OUTPUT
```

Notes

Example 3 appends the contents of file 'bbb' to file 'aaa.' Notice that the last line of the program CLOSEs the files that OPEN has opened. This practice is strongly recommended: you should always close a file when you have finished writing to it.

See Also

?, CLOSE, INPUT, OUTPUT TO

OPEN COMMS

Purpose

Opens the communications channel.

Syntax

OPEN COMMS [USING *parameters*]

Comments

This command must be executed before attempting file or text transfer. USING can take up to thirteen parameters, each separated by a comma. Most of the parameters correspond to those in the comms requester; parameters 3 to 5 provide options which are not available from the requester, and are useful when you are setting up the comms facility for 'chat mode.'

param1: Unit number (0 for internal serial port, or unit number for required port on multiple serial card)

param2: Baud rate (300/1200/2400/4800/9600/19200)

param3: Parity (0/1/2)

param4: Stop bit (either 1 or 2)

param5: Word length (5/6/7/8)

param6: Transmit/Receive (1 for Transmit, 0 for Receive)

param7: File Header (0 for Off, 1 for On)

param8: WXMDEM (0 for Off, 1 for Use)

param9: CRC-XM (0 for Off, 1 for Use)

param10: Chop (0 for Off, 1 for Use)

param11: LF/CR (0 for Conversion, 1 for No conversion)

param12: Autodial number (text string). Note that Superbase provides the 'ATD' sequence automatically.

param13: Initialization sequence (up to 28 characters, within quotes)

The Autodial number and Initialization sequence must be supplied as a text string; the other parameters must be numeric values. If you do not supply these parameters, Superbase uses the values currently shown in the comms requester. These default values are stored in the S:SUPERBASE.INI file when you click on OK in the requester.

It is important to realize that OPEN COMMS does not initiate any activity other than opening the communications channel and setting the comms options. For instance, if you have supplied an Autodial number, executing the OPEN COMMS command does not actually dial the number; this would be done if you issued a COMMS FILE ? command and DCD was not active.

For simple two-way communication do not specify a dial string, use instead a COMMS ? command to send the required string to the modem, as illustrated in Example 2 below.

OPEN COMMS also has the effect of setting the options in the comms requester; i.e. if you select the WXMODEM protocol with OPEN COMMS, this option will be highlighted when you next display the comms requester.

Examples

```
1 OPEN COMMS USING 1, 1200, 0, 1, 8, 1, 1, 0, 0, 0, 0, "12149264521", ""
```

Here the command opens a channel for remote transmission to a modem on the telephone number 1 214 926 4521.

```
2 OPEN COMMS USING 1, 1200, 0, 1, 8, 1, 1, 0, 0, 0, 0, "", ""  
COMMS ? "ATDT12149264521"
```

This immediately dials the specified number and establishes communication.

See Also

CLOSE COMMS, COMMS ?

OPEN FIELDS

Purpose

Specifies which fields are displayed.

Syntax

```
OPEN FIELDS [FILE sbfname] [fieldlist] / [ADD fieldlist]
```

Comments

This command is the program equivalent of the Field Selection command on the Set menu.

When used on its own or with the FILE parameter, OPEN FIELDS activates the file's field list – the field list for the current file is saved using the Save File option (i.e., it is stored with the file definition), and it is loaded from disk (but not activated) when the file is opened. The field list does not take effect until the current record is redisplayed.

To specify under program control which fields are to be open, use the *fieldlist* parameter. It should consist of a list of field names separated by commas.

Use the ADD parameter to add other fields to the existing set.

To remove any restrictions on which fields are shown, use the CLOSE FIELDS command.

Examples

- 1 OPEN FIELDS FILE "Address" Firstname, Lastname, Country, City
- 2 OPEN FIELDS: VIEW
- 3 OPEN FIELDS ADD Code

See Also

CLOSE FIELDS

OPEN FILE

Purpose

Opens a database file and its default index.

Syntax

OPEN FILE *sbfname* [:password] / *dbfname.dbf*

Comments

Note the distinction between OPEN FILE "aaa" which opens a database file, and OPEN "aaa" (FOR INPUT) which opens a text file.

sbfname is compulsory, and if a password is required to access the file, then it is also compulsory (use a semicolon to separate the filename from the password).

INDEX followed by a field name may be added to the end of an OPEN FILE command, allowing you to select an index other than the default index. But it can only be used if the file has already been opened by a direct command or an earlier program line. As explained in the entry for the INDEX command, DML parses the whole line before executing it; so if you refer to a field, it must be a field in a file that has already been opened. Otherwise an error will result.

To open a dBase file, specify the DBF extension explicitly.

Examples

- 1 OPEN FILE "aaa"
- 2 x\$="bbb": OPEN FILE x\$
- 3 OPEN FILE "aaa;John"

Notes

In example 3, 'John' is the password for the file 'aaa'.

See Also

CLOSE FILE, FILE, SHARE

OPEN FORM

Purpose

Loads a form from disk and displays it in the database window.

Syntax

OPEN FORM *form* [PAGE *nexpr*]

Comments

form must be a string expression giving the file name of a form. The keyword PAGE and its following numeric expression allow you to specify that a specific page of the form should be displayed when the form is opened. If used, the result of *nexpr* must be a valid page number. Superbase will also open any database files associated with the form. To specify which part of the form is displayed in the Superbase window use the FORM command.

OPEN FORM always reads the SBV file from disk. To avoid this, you can use the FORM system variable to test whether the required form is already open.

Examples

- 1 OPEN FORM "Invoice"
- 2 IF FORM <> "Invoice" THEN OPEN FORM "Invoice"

See Also

BLANK FORM, CLOSE FORM, CREATE FORM

OPEN INDEX

Purpose

Open an dBase NDX file for use with a dBase DBF file.

Syntax

OPEN INDEX *dbasefile*.NDX

Comments

The command allows you to select a dBase index file for use in dBase browsing and reporting activities. See Volume 1, Chapter 11 Working with dBase Files.

Examples

```
1 OPEN INDEX "account1.ndx"
```

Opens the account1.ndx file, which is created within the dBase program, as the current index for the dBase file.

See Also

INDEX, OPEN FILE

ORDER

Purpose

Sets the order for Query output. Also, as a system variable, returns the data entry order of the field most recently edited on a form.

Syntax

ORDER [ASK] / [[&*nexpr*] *field* [ASCENDING/DESCENDING]
[,*field*] ASCENDING/DESCENDING] [,.....]

Comments

ORDER as a system variable. When used as a system variable, ORDER allows you to determine the data entry order of the last field edited on a form.

Normally the ORDER system returns the data entry order number of the last field or variable entered during Form data entry. However, if the user presses ESCAPE to terminate data entry, ORDER returns a negative number.

If data entry is being done in Page or Record view, there is no data entry order number, as this feature only applies to Forms. ORDER returns 1 when data entry ends after the last field has been completed, or -1 if ESCAPE is used to terminate data entry.

ORDER as a DML command. ORDER is a Query Language command and can only be entered in a query section – i.e., it works in conjunction with the Query Language command SELECT.

This command is the program equivalent of the Order command line in the Query Definition requester. If you follow ORDER with the ASK parameter, you will be presented with the Order requester. This is the same as the requester that appears when you click on the Order button from the Query Definition requester. You can then define the Order line in the same way as you would within Superbase – i.e. by clicking on the various items within the requester.

If the user clicks the Cancel button on an ORDER ASK requester, a CTRL+C error is generated (error 11). This should be trapped and handled with the ON ERROR, ERRNO, ERR\$, and RESUME group of commands.

If you do not use the ASK parameter, you will need to follow the command with at least one field name.

The field specified with the ORDER command determines the order in which the fields in the SELECT line are output. If you are familiar with the concept of sorting, you can think of ORDER as setting the sort 'key' for query output.

The default length for sorting is 15 characters per field. Superbase gives equal weighting to upper case, lower case and accented instances of characters. The & character followed by a value up to the length of the field may precede any field, specifying the number of characters that will be used in sorting.

field must be a field in an open file, but it does not need to be an indexed field; nor does it have to be one of the fields in the SELECT line.

ASCENDING and DESCENDING allow you to specify whether data is sorted in ascending or descending order. If you specify a text field with the ORDER command – i.e., if you specify it as the sort key – Superbase outputs record data according to the alphabetical order of the sort field. DESCENDING reverses the order and sorts the field from Z to A.

With numeric, date and time fields, ASCENDING sorts data in numeric, date or time order; and DESCENDING reverses the order.

By default, fields are sorted in ascending order; so it not strictly necessary to include the ASCENDING parameter.

You can also specify more than one field in the ORDER line, separating each with a comma. If you enter two fields, the first field takes precedence as a sort key over the second field: i.e., records are first sorted according to the first field, and then any duplicate data items are sorted according to the second key.

The same applies if there are more than two fields: the second key has priority over the third, the third over the fourth, and so on.

Using a Semicolon with SELECT

It is possible to suppress the output of a multi-line SELECT command by using a semicolon:

```
SELECT ;
```

This command will read all the records in the current file without outputting any data.

The feature may be used in conjunction with BEFORE SELECT or AFTER SELECT when special processing is required.

Examples

The examples illustrate how ORDER works by taking a limited set of records and showing some of the different ways in which they may be sorted. Each record contains data from three fields, Firstname, Lastname, and Country.

```
1  SELECT Firstname, Lastname, Country
   ORDER Lastname
```

This example takes Lastname as the sort key and produces the following output:

Firstname	Lastname	Country
Pierre	Arnauld	France
Robert	Brown	England
William	Carter	USA
Gerde	Hemrich	West Germany
John	Miles	England
Anne	Richardson	USA
Peter	Smith	England

```
2  SELECT Firstname, Lastname, Country
   ORDER Country
```

The output from this query would be as follows:

Firstname	Lastname	Country
Robert	Brown	England
John	Miles	England
Peter	Smith	England
Pierre	Arnauld	France
William	Carter	USA
Anne	Richardson	USA
Gerde	Hemrich	West Germany

- 3 `SELECT Firstname, Lastname
ORDER Country DESCENDING, Lastname ASCENDING`

The output from this query is:

Firstname	Lastname
Gerde	Hemrich
William	Carter
Anne	Richardson
Pierre	Arnauld
Robert	Brown
John	Miles
Peter	Smith

- 4 `SELECT Firstname, Lastname, Country
ORDER Country, Firstname`

This example uses Country as the primary sort key and Firstname as the secondary key to produce the following output:

Firstname	Lastname	Country
John	Miles	England
Peter	Smith	England
Robert	Brown	England
Pierre	Arnauld	France
Anne	Richardson	USA
William	Carter	USA
Gerde	Hemrich	West Germany

- 5 `x%% = ORDER`

This example stores the data entry order of the last field to be edited (on a form) in the numeric variable x%%.

See Also

SELECT GROUP

OUTPUT TO

Purpose

Opens a text file on disk for output.

Syntax

OUTPUT TO *filename*

Comments

This command makes the disk the current output device and sends any future output to *filename*. It takes the current page format into account and inserts line and page breaks in the file as they occur on screen. In other words, this command creates a disk image of the output to a printer.

To create a sequential ASCII file, use the OPEN FOR OUTPUT command.

If the text file already exists on disk, any output command issued after OUTPUT TO, will overwrite the file. If you want to add data to an existing text file, use OPEN *filename* FOR APPEND.

Examples

- 1 OUTPUT TO "Names"
 ? Lastname
 CLOSE OUTPUT
- 2 OUTPUT TO a\$

Notes

Example 1 stores the contents of the Lastname field (in the current record) on disk in the text file Names.

See Also

?, CLOSE OUTPUT, INPUT, OPEN FOR

PAD\$

Purpose

Defines the length of a string variable or text field.

Syntax

PAD\$(*strexpr* [,*nexpr*])

Comments

When you create a composite index using a calculation formula that concatenates fields together, you may need to ensure that each element in the resulting key has a fixed length. PAD\$ truncates a string or extends it with spaces to achieve this.

strexpr may be a string variable or text field.

nexpr specifies the total number of characters that you want. The string is truncated or lengthened with spaces if necessary so that it becomes the length you want.

If you omit *nexpr*, then a field is padded to the length specified in the file definition, while a string variable remains unchanged.

Examples

1 PAD\$(City.Clients) + Lastname.Clients

The result of the expression will be a string in which the City element is always the same length, whether the actual city is York or Washington.

Notes

See Volume 1, Chapter 4 Creating Indexes for an extended example of how to use PAD\$.

See Also

SPACES\$

PANEL

Purpose

Switches the control panel on or off.

Syntax

PANEL ON/OFF

Comments

This command is provided for people who are developing Superbase applications; for example, programmed applications for use with the Superbase run-time system. When a program is running, the control panel is ghosted. You may turn it on using the WAIT PANEL command; but if you decide that the user should not have access to the buttons on the control panel, you may prefer to remove it altogether by executing the command:

PANEL OFF

PANEL ON then returns the control to the screen.

See Also

WAIT

PASSWORD

Purpose

Sets new passwords (or none) for a specified file.

Syntax

PASSWORD *sbfname* [;passwords]

Comments

sbfname must be an open file and, as usual with filenames, must be included in quotation marks.

If no password is given, the existing password for the specified file is removed.

Examples

```
1 OPEN FILE "aaa;John"  
  PASSWORD "aaa"
```

Removes passwords.

```
2  PASSWORD "aaa;Rosebud"
```

Sets a password for the file 'aaa'.

```
3  OPEN FILE "aaa;John"  
    PASSWORD "aaa;John;Paul;George"
```

Adds passwords for read/write and read only access privileges.

See Also

OPEN FILE

PCOL

Purpose

Return the column position of the print head on the current output printer or resets the print head's position.

Syntax

```
PCOL ( nexpr )
```

Comments

If *nexpr* is zero, the function returns the column position of the print head on the current printer. For the Row position, see PROW. See also LOCATE.

You can also use this function to set the counter Superbase uses to keep track of the print head's position. Giving *nexpr* a positive value, sets the counter to that value. The print head itself is not moved. This feature is used to reset the internal count after issuing a series of printer commands which have not in fact moved the print head, for example, after switching to high density graphics mode.

Examples

```
1  x%% = PCOL(0)
```

```
2  ? PCOL(0)
```

See Also

PROW

POSITION

Purpose

Returns the position of the data pointer in an ASCII file or sets the pointer to a new position.

Syntax

POSITION [*nexpr*]

Comments

When you read data from an ASCII file on disk, Superbase uses an internal pointer to keep track of it. The OPEN *file* FOR INPUT command sets the pointer to zero, the position of the first character in the file. Thereafter it is incremented by one for each character that is input using the INPUT command.

POSITION may be used in two ways. Firstly, it returns the position of the data pointer when an ASCII file is being read with the INPUT command. Secondly, POSITION sets the pointer to the character position specified by *nexpr*.

Normally, the data in an ASCII file is read into the computer sequentially. With POSITION, you can input character data on a more selective basis.

You will only be able to take advantage of this command if you know where the data is stored in a file. Superbase stores data in variable length fields (as opposed to fixed length fields): when you create an ASCII file from an SBF file by exporting it, the amount of space occupied on disk by field data may vary from record to record. This means that there is no simple way of knowing the position of any particular field or record.

One solution to this problem is to create an ASCII file from a database file using the query option Output to Disk. When you do this, Superbase stores the data in fixed length fields – each field takes the length set in the file definition. You can then work out the number of characters occupied by a record in the ASCII file and use this figure to retrieve specific records or fields. For example, if the record length was 49 characters, you would enter:

```
POSITION 49 * 5: INPUT LINE a$
```

to retrieve the fifth record in the file.

Examples

```
1  OPEN "Cust.asc" FOR INPUT  
   FOR n&% = 0 TO 76 * 12 STEP 77
```

```
POSITION n&%  
INPUT &15, a$  
? a$  
NEXT  
CLOSE INPUT  
2 IF POSITION > nchar%% * 10 THEN ...  
3 CLS                                OPEN "cl.asc" FOR INPUT  
  INPUT LINE y$  
  z%% = LEN(y$) + 2:      REM Allows for CR/LF  
  CLOSE INPUT  
  getrec:  
  ASK "Enter record number: ";n&%  
  IF n&% 1 THEN END:      REM Enter 0 to stop  
  OPEN "cl.asc" FOR INPUT  
  POSITION z%% * (n&% - 1): REM To get 1st rec if you enter 1  
  INPUT LINE x$  
  ? &4.0 LEN(x$); &8.0 POSITION  
  ? x$  
  CLOSE INPUT  
  GOTO getrec
```

Notes

The first example inputs the first field from the first twelve records in the ASCII file Cust.asc. It assumes that the record length is 77 characters and that the length of the first field is 15 characters. In the second example, POSITION is used to return the pointer position.

The third example prompts for a record number, reads that record from a file and displays the contents. It should work with any ASCII sequential file of fixed-length records.

See Also

INPUT

PRINT

Purpose

Sends information to the printer.

Syntax

PRINT CURRENT [PAGE/ALL [USING *nexpr1* ... *nexpr7*]] / FILE [PAGE/ALL [USING *nexpr1* ... *nexpr7*]] *sbfname* [INDEX *indexname*] [DESCENDING] [WHERE *conditions* / ASK] / [*expressionlist*]

Comment

PRINT can be used to print data from variables, or with the FILE or CURRENT options to print record or form data.

PRINT CURRENT outputs the current record in Record or Page View.

If you choose the FILE option, PRINT outputs data from the specified file in the format of the current Record, Page, or Table View.

If a form is open, all the data on the form will be output. The ALL parameter causes all the pages in a multi-page form to be printed; PAGE prints only the current page.

PAGE / ALL may be followed by USING with up to seven optional parameters to specify centering and object class selection. Each parameter may have the value 1 or 0 to signify whether the object class should or should not be printed. The parameters may be numeric expressions. The order of the parameters is as follows:

- | | |
|---|-----------|
| 1 | Centering |
| 2 | Areas |
| 3 | Boxes |
| 4 | Lines |
| 5 | Text |
| 6 | Images |
| 7 | Fields |

If all parameters are set to 0, the default requester is presented so that the user may choose desired options.

The INDEX and optional DESCENDING keywords allow you to specify an index other than the current index, and to choose descending sorting order.

A WHERE clause selects records from the current file using a filter. If a file involved in a multi- file form is not the current file but has had a filter set previously, that filter will be also be used to select records.

When printing starts, screen and keyboard activity pauses, as indicated by a change in the appearance of the mouse pointer.

To end printing, you must execute a `DISPLAY` command to re-enable the Superbase window. This causes a form feed on the printer, so it is advisable to group `PRINT` statements together and not interleave them with `DISPLAY` statements.

`PRINT`, followed by a semicolon and nothing else, selects the printer as the current output device. The `?` command can then be used to send information to the printer. You can also use `PRINT` to output information directly to the printer, by following the command with one or more expressions. But note that any use of `PRINT` makes the printer the current output device.

The items in an expression list following the `PRINT` command may be separated by a semicolon or a comma. If a semicolon is used, Superbase will print the expressions one after the other without any spaces in between; a comma has the effect of inserting a space between items. In some circumstances, you may also dispense with separators altogether. Thus, provided it can distinguish between different items, Superbase will accept a list of expressions which are entered on the line head to tail; for example:

```
PRINT a$b$c%"Hello"
```

Examples

- 1 `PRINT;`
`? MEMORY`
`DISPLAY;`
- 2 `PRINT UL "The items in the following list will be printed underlined"`
`PRINT "One","Two","Three";`
`DISPLAY;`
- 3 `PRINT FILE ALL "accounts" INDEX Balance DESCENDING WHERE`
`Balance > 5000`
- 4 `PRINT CURRENT PAGE USING 1,0,0,0,1,0,1`

Notes

The first example prints the current program's variables and their contents. Example 2 prints a list of items. Both examples use the `DISPLAY` command to make the screen the current output device after the print operation is finished. Example 3 prints all pages for every account record with a balance greater than \$5000, starting with the largest. Example 4 prints the current page centered, showing only text and field data.

See Also

`DISPLAY`

PRINTER

Purpose

Returns the current printer name.

Syntax

PRINTER

Examples

```
1  REQUEST "Your current printer is " + PRINTER,"Change it?",127,x%  
   IF x% THEN ...
```

See Also

SET PRINTER, SET PRINTER REQUEST

PROTECT

Purpose

Saves the current program in an encrypted form.

Syntax

PROTECT *filename*

Comments

Use this command to ensure that program files are not seen by anyone else. It stores a file on disk in an encrypted (scrambled) form so that it can be run but not edited.

If the first line of a program is a REM statement, PROTECT displays that line, but hides the rest of the program from any attempt to edit or LIST it.

Examples

```
1  PROTECT "myprog"
```

See Also

LIST, LOAD, SAVE

PROW

Purpose

Returns the row position of the print head on the current output printer.

Syntax

PROW (*nexpr*)

Comments

If *nexpr* is zero, the function returns the row position of the print head on the current printer. For the Column position, see PCOL. See also LOCATE.

Giving *nexpr* a positive value resets Superbase's internal row counter. See PCOL.

Examples

```
1  x%% = PROW(0)
```

```
2  ? PROW(0)
```

See Also

EJECT, PCOL

QUIT

Purpose

Exits from Superbase.

Syntax

QUIT

Comments

This has the same effect as selecting the Quit command on the Project menu. It exits from Superbase and returns the user to Workbench.

See Also

END

READ

Purpose

Reads the data given in a DATA statement and assigns it to a variable or field.

Syntax

```
READ var / field [var / field] [...]
```

Comment

The types of variables or fields used with a READ command must match the types of data expected – numeric variables or numeric fields for numeric data and string variables or string fields for string data.

DML uses a pointer to keep track of where it is in the list of DATA items; that is, each time a data item is read, DML moves the pointer on to the next item in the list. If you wish to read the same data again, you can place a label in front of a DATA statement and use RESTORE.

Examples

```
1  READ a#%, b$, fielda.filea, fielda.fileb
2  DIM a#%(19)
   FOR n%% = 0 TO 19
   READ a#%(n%%)
   NEXT
```

See Also

DATA, RESTORE

RECCOUNT

Purpose

Counts the number of records in a file, or the number of transaction records retrieved during a SELECT FORM operation.

Syntax

```
RECCOUNT ( sbfname ) / ("*)
```

Comments

This function returns a number showing how many records there are in the file specified. You can use the empty string as an argument – RECCOUNT ("") – to refer to the current file.

RECCOUNT ("*") is used to determine whether there are any transaction records remaining to be retrieved when browsing with a one-to-many relational form. If RECCOUNT ("*") returns 0, there are no more records to be retrieved.

Examples

```
1  ? RECCOUNT("Orders")
2  x&% = RECCOUNT(x$)
3  OPEN FILE ("address")
   SELECT FIRST: VIEW
   FOR n&% = 1 to RECCOUNT("address")
   SELECT NEXT
   VIEW
   NEXT n&%
4  IF RECCOUNT ("*") <> 0 THEN
   SELECT FORM NEXT PAGE
   ELSE
   SELECT FORM NEXT
   END IF
```

Notes

Example 3 displays all the records in the file 'address' in turn. Example 4 steps through a master file retrieving all transactions page by page before moving to the next record.

See Also

SER

REM

Purpose

Inserts a non-executable comment (a remark) into a program.

Syntax

REM [*text*]

Comment

REM has the effect of canceling any statements after it. This makes it useful when you are testing a program – placing it at the start of a multi-statement line puts the

following statements temporarily out of action. More generally, use REM to annotate a program in order to explain how it works or what it does.

A single quotation mark after a command without an intervening colon also acts as a REM statement.

Examples

- 1 REM This is a remark
- 2: FILE "aaa" 'open aaa
- 3: FILE "aaa": REM open aaa
- 4 FILE "aaa": REM eliminate next commands: INDEX abc: SELECT FIRST

Notes

Examples 2 and 3 have identical effects and demonstrate the two different ways of entering a comment. In example 4, the REM statement means that the INDEX and SELECT FIRST commands are not executed.

REMOVE FILE

Purpose

Removes a database file from disk, along with its associated definition and index files.

Syntax

REMOVE *sbfname*

Comment

This command operates in the same way as the Project Remove menu command. Note that you are not asked for confirmation – the file is just removed.

Examples

- 1 REMOVE FILE "aaa"
- 2 REMOVE FILE "dh0:sbase/aaa"

See Also

ADD, CREATE, DELETE, MAKE

REMOVE FROM

Purpose

Removes records which match the conditions specified.

Syntax

REMOVE FROM FILE *sbfname* [WHERE *conditions*]

Comment

This command works in the same way as the equivalent Remove command on the Process menu. It deletes records from a file on disk.

FILE *sbfname* has to be open, and if the file requires a password, you must have full access to it.

WHERE *conditions* is optional and is set up in the same way as a filter. If it is not included, the command acts on all the records in a file.

Examples

- 1 REMOVE FROM FILE "aaa" WHERE Lastname LIKE "[a-c]*"
- 2 REMOVE FROM FILE "aaa"

This empties the file of all its data.

See Also

CREATE FILE, REMOVE, SELECT

REMOVE INDEX

Purpose

Removes an index on the current file from disk.

Syntax

REMOVE INDEX *index*

Comment

This command works in the same way as the Remove Index option on the Project menu. The file must be open, and, if it requires a password, you must have full access privileges.

index is the name of an indexed field. It can be entered with a file extension.

Examples

- 1 REMOVE INDEX fielda
- 2 REMOVE INDEX fieldb.aaa

See Also

CREATE INDEX, INDEX

RENAME

Purpose

Renames a file on disk.

Syntax

RENAME *old.filename* [, / TO] *new.filename*

Comments

This command works in the same way as the RENAME option in Workbench, but allows you to rename a file without exiting from Superbase. You have the option of using either a comma or the keyword TO as the separator between the two file names.

Examples

- 1 RENAME "aaa", "bbb"
- 2 RENAME "aaa" TO "bbb"

See Also

REORGANIZE

REORGANIZE

Purpose

Reorganizes the current file or a specified file.

Syntax

REORGANIZE [FILE *sbfnamea*] [TO] *sbfnameb*

Comments

This command is the program equivalent of the Reorganize command on the Process menu. It takes a file on disk, reorganizes it, and stores it as *sbfnameb*. If the FILE option is not used, the current file is reorganized.

sbfnameb can include the pathname for another directory or disk. If you enter a pathname without a file name following it, the file will be reorganized under the same name.

Examples

- 1 FILE "aaa": REORGANIZE TO "copy"
- 2 REORGANIZE FILE "aaa" TO "dh0:mydir/aaa"

Notes

Example 1 creates a reorganized file 'copy' in the current directory. Example 2 creates a file 'aaa' in the directory 'mydir'.

See Also

COPY, CREATE ... FROM ... WHERE, RENAME, SELECT TO

REPLICATE

Purpose

Replicates a string a given number of times.

Syntax

REPLICATE (*strexpr*, *nexpr*)

Comments

REPLICATE repeats the character string *strexpr* the number of times given by *nexpr*.

Examples

- 1 textfieldc = REPLICATE("*-+-", 10)
- 2 x\$ = REPLICATE(textfieldc, 4)
- 3 x\$ = REPLICATE (" ", 25)

Notes

Example 3 sets x\$ to 25 spaces, but see the function SPACE\$.

See Also

SPACE\$

REPORT**Purpose**

Specifies the field or fields on which totals (and other report statistics) will be produced for the report as a whole.

Syntax

REPORT [SUMMARIZE] [*params*] *fieldname* [*,fieldname*] [...]

Comment

Report has two uses. When you create a Report with the Form Designer, Superbase generates a Report statement by noting the fields which have been specified in an AFTER REPORT section; i.e., if the AFTER REPORT section in a Report program contains the statements:

- ? SUM amount
- ? COUNT deposits

Superbase will generate the following line in the program:

REPORT amount, deposits

If you are writing a Report program yourself (as opposed to modifying a program generated by Superbase), you should remember to enter a Report statement including the names of any fields for which you wish totals and other report statistics to appear.

The second application for REPORT is as a query language command. In this context, it allows you to create a program line which is equivalent to the REPORT command line in the query definition requester.

REPORT is used here to specify the fields for which totals and other statistical data are requested.

When you use the SUMMARIZE option, Superbase suppresses the main detail of the report and prints just the summary information.

See Also

AFTER REPORT, BEFORE REPORT, END REPORT, FOOTING, HEADING

REQUEST

Purpose

Displays a Superbase requester.

Syntax

REQUEST *text1,text2,type[,nvar[,strvar[,len]]]*

Comment

REQUEST allows you to select one of Superbase's requesters and display it on screen. There are two groups of requesters, which differ slightly in their appearance.

To some extent you can customize a requester to your own requirements. You can place a title in the box, and in some cases specify the text string that initially appears in the requester's command line or selection box. For certain requesters, it is also possible to specify the length of the box.

text1 and *text2* are the first and second line of the requester title. They must be included although they can be "". The maximum length for each line is 50 characters.

type is the requester type. It defines the type of requester according to the table shown below.

nvar is a numeric variable. It returns a value of 1 if OK is selected and there is an entry into the string requester. If CANCEL is selected or there is no entry into the string requester, it returns 0. *nvar* is optional and can be replaced by a comma.

strvar can be used with requesters which have a string entry box and has two functions:

- It is used to place a default value into the string box, i.e., the text string in *strvar* is entered into the string box when the requester is displayed.
- It returns the string which the user enters in the box.

len specifies the length of the string box (where appropriate). This is particularly useful for the information requesters.

Group 1 Request Requesters

These requesters are visually the same as Superbase's own requesters.

Type	Requester	Buttons
0	string	OK
1	string	OK CANCEL
2	string	OK
3	string	OK CANCEL
4	string	OK CLEAR CANCEL
5	Current file fields	OK CLEAR CANCEL
6	Open Fields list	OK CLEAR CANCEL
7	Indexed fields	OK CLEAR CANCEL
8	Non indexed fields	OK CLEAR CANCEL
9	Field Info	OK CLEAR CANCEL
10	Open Database Files	OK CLEAR CANCEL
11	Database Files	OK CLEAR CANCEL
12	Program Files	OK CLEAR CANCEL
13	Text Files	OK CLEAR CANCEL
14	Query Files	OK CLEAR CANCEL
15	Update Files	OK CLEAR CANCEL
16	Function Key Files	OK CLEAR CANCEL
17	Editable *.* File Specification	OK CLEAR CANCEL
18	Pre-specified File Specification	OK CLEAR CANCEL
19	Form Files	OK CLEAR CANCEL
20	Record or Array List Selection	OK CLEAR CANCEL
21	Same as requester 1	
22	Same as requester 2	
23	Same as requester 4 but does not echo keyboard input	
24	Same as requester 20, but uses the current filter setting to select the records that are shown in the dialog.	

- A database file must be open before types 5 to 9 can be selected.
- Types 2 and 3 are displayed for a fixed period, then removed automatically.
- Types 21 and 22 are only maintained for compatibility with earlier versions of Superbase.
- Type 11 limits the visible length of the first string to about 20 characters, depending on resolution. If the dBase checkbox is turned on, then Superbase adds a DBF extension to the selected file name.
- Type 18 allows the return string variable to be used to preset the file specification using wildcards. The selected filename is held in the return variable.

Positioning Co-ordinates. REQUEST types 0 through 24 may be positioned within the work area by placing parameters of the form *@nexpr_x,nexpr_y* before the *text1* string, where *nexpr_x* designates the x dimension and *nexpr_y* designates the y dimension. Each expression contains a value in pixels.

REQUEST types 100-151 cannot be positioned in this way.

REQUEST 20

Requester type 20 may be used in conjunction with the LOOKUP function for validating entered data against the contents of another file. It may also be used for listing array contents. REQUEST type 20 has a different syntax to the other types:

REQUEST *text1*, *text2*, *type*, [*nvar*], *field1*/*var*, *len*, *field2*/*arrayvar*,
[*field3*/*arrayvar* [*field4*/*arrayvar*]]

Selecting from a file

The first two parameters have the same function as with other requester types: they give the first and second lines of the requester title. *type* can be any numeric expression with a value of 20. *nvar* is a numeric variable which returns 1 if OK is clicked, 0 for Cancel; Cancel also generates error number 11 (CTRL+C).

The next parameter – *field1* / *var* – may be a field or a variable. If a field, it will usually be the name of a field in the current file but it can also be a field in another open file. Whatever is entered in the requester is stored in the field or variable. The fields specified with *field* parameters 2 to 4 can also belong to any open file but it must be the same file. *len* specifies the width of the requester in number of characters.

When you execute this command, it lists the contents of *field2* for each record in the file. If *field3* and *field4* are specified, the contents of these fields will also be displayed in the requester to the right of *field2*.

Superbase retrieves the *field2* data using the current value of *field1* or *var* as an index key or a 'seed.' Provided *field2* is an indexed field, the list of fields in the requester will start with the field before the one that matches this key. If it is not indexed, the field data is listed in the order of the current index.

When you select one of the *field2* values from the requester and click OK, Superbase stores it in the field or variable specified with *field1* / *var*.

Selecting from an array

The purpose of this facility is to allow an application to set up a list of text or numbers from which users can make a selection. For example, a field might contain one of a range of twelve country names. Rather than store the country names in a table file, you could place them in an array, and use this command to enable the user to pick one during data entry.

The contents of the array must be defined prior to the execution of the command. When Superbase executes the command, it lists the full contents of the array. The *arrayvar* must be specified without any parentheses. Both string and numeric arrays may be used, but only the contents of the first dimension of a multi-dimensional array will be listed.

If more than one *arrayvar* is used, Superbase concatenates each line in the list, so a

tabular format will not appear unless all the elements in the array are the same length.

Group 2 Request Requesters

Group 2 requesters have a different style of title bar.

Type	Buttons
100	OK
101	OK Cancel
102	Abort Retry Cancel
103	Yes No Cancel
104	Yes No
105	Retry Cancel
106	OK Cancel
107	Abort Retry Cancel
108	Yes No Cancel
109	Yes No
110	Retry Cancel
111	Abort Retry Cancel
112	Yes No Cancel
113	OK
114	OK Cancel
115	Abort Retry Cancel
116	Yes No Cancel
117	Yes No
118	Retry Cancel
119	OK Cancel
120	Abort Retry Cancel
121	Yes No Cancel
122	Yes No
123	Retry Cancel
124	Abort Retry Cancel
125	Yes No Cancel
126	OK
127	OK Cancel
128	Abort Retry Cancel
129	Yes No Cancel
130	Yes No
131	Retry Cancel
132	OK Cancel
133	Abort Retry Cancel
134	Yes No Cancel
135	Yes No
136	Retry Cancel
137	Abort Retry Cancel

138	Yes No Cancel
139	OK
140	OK Cancel
141	Abort Retry Cancel
142	Yes No Cancel
143	Yes No
144	Retry Cancel
145	OK Cancel
146	Abort Retry Cancel
147	Yes No Cancel
148	Yes No
149	Retry Cancel
150	Abort Retry Cancel
151	Yes No Cancel

The syntax for all Group 2 request types is the same:

REQUEST *text1,text2,type[,nvar]*

The types are categorized into four groups, with each group subdivided according to which button is the default for the ENTER key.

Icon

Types 100-112 display an 'i' (for information) icon.

Types 113-125 display an exclamation point icon.

Types 126-138 display a question mark icon.

Types 139-151 display a STOP icon.

Default Button

Types 100-105, 113-118, 126-131, and 139-144 take the left-hand button as the default.

Types 106-110, 119-123, 132-136, and 145-149 take the second button from the left as the default.

Types 111-112, 124-125, 137-138, and 150-151 take the third button from the left as the default.

Value Returned in *nvar*

Cancel or Abort return 0.

OK or Yes or Retry return 1.

Ignore returns 2.

No returns 0 unless the requester includes a Cancel, in which case it returns 2.

Examples

```
1 REQUEST "", "", 2, a%
```

Shows a Group 1 style requester with an OK button only for a short time, then removes it without user action being required.

```
2  REQUEST "Exit Application", "", 129, a%
```

Presents a Group 2 style requester with a question mark icon, Yes, No, and Cancel buttons, and a default Yes.

```
3  a%=0: a$="": REM initialize nvar and strvar
   REQUEST "Select a program", "", 5, a%, a$, 32
   IF a%=0 THEN ? "No program selected!": END
   CHAIN a$
```

Allows the user to select a program to run, and terminates if the user clicks Cancel.

```
4  REQUEST "Select a customer code", "", 20, a%, Cuscode.Invhdr, 40,
   Cuscode.Customer, Firstname.Customer, Lastname.Customer.
```

Illustrates the use of requester type 20. Taking data from the **Customer** file, the requester lists the names and customer codes of the individuals in the file. When the user selects a customer code from the requester, it is stored in the **Cuscode** field in the current record from the **Invhdr** file.

RESTORE

Purpose

Moves the data pointer back to the first DATA statement, or to a specified label.

Syntax

```
RESTORE [label]
```

Comments

The data pointer is the internal pointer that Superbase uses to keep track of which DATA statements have been read. Initially it has a value of zero and points to the first DATA statement. As you READ data into variables or fields, the data pointer is increased by one for every data item read.

This command resets the data pointer. If you do not specify *label*, the data pointer is reset to the beginning of the first DATA statement. If you specify *label*, the data pointer is reset to the data statement following the label.

Examples

```
1  RESTORE
```

```
2  RESTORE label1
```

See Also

DATA, READ

RESUME

Purpose

Resumes execution after an error.

Syntax

```
RESUME [NEXT/label]
```

Comments

The RESUME command works in conjunction with the ON ERROR GOTO command which is used to trap program errors.

RESUME, on its own, returns program control to the statement that caused the error. When NEXT is included, the statement returns program control to the statement after the one which caused the error.

label transfers program control to the label specified.

Examples

```
1  REM Top of program
   ON ERROR GOTO errlab1
   SELECT FIRST:? fieldname
   .....
   .....
   errlab1: IF ERRNO=44 THEN OPEN FILE "aaa":REM file not open
   RESUME

2  REM read data
   ON ERROR GOTO errlab2
   .....
   READ x$
   .....
   endread:.....
   .....
```

.....
errlab2: IF ERRNO=18 THEN RESUME endread:REM out of data

Notes

The above examples deal with only certain 'expected' errors. Complete programs should deal suitably with all possible errors.

See Also

ON ERROR GOTO

RETURN

Purpose

Returns from a subroutine.

Syntax

RETURN

Comments

The RETURN command is used to mark the end of a subroutine. It instructs the computer to transfer program control to the statement immediately following the GOSUB or ON GOSUB statement which initially called the subroutine. See GOSUB.

See Also

GOSUB, ON ... GOSUB

RIGHT\$

Purpose

Extracts the rightmost one or more characters from a text string or text field, or transfers spaces to the beginning of the string.

Syntax

RIGHT\$ (*strexpr*, [*nexpr*])

Comments

If *nexpr* is specified, this function extracts the rightmost *nexpr* characters from *strexpr*.

If *nexpr* is omitted, a string is returned of the same length as *strexpr* and containing the same characters, but with any trailing spaces transferred to the beginning.

Examples

```
1  textfieldc = RIGHT$(textfielda, 10)
2  textfieldc = LCASE$(RIGHT$(textfielda, 1))
3  IF RIGHT$(textfielda, 1) = "s" THEN ...
4  x$ = RIGHT$("ABCD ", 5)
5  x$ = RIGHT$("ABCD ")
```

Notes

Example 4 sets x\$ to "CD ", while Example 5 sets it to " ABCD".

See Also

LEFT\$, LEN, MID\$

RND

Purpose

Returns a random number.

Syntax

RND (*nexpr*)

Comments

What the function returns depends on the value of *nexpr*.

If *nexpr* is less than zero, the random number generator is reseeded. This means that a new series of random numbers will be generated, completely unrelated to the last series. It also allows you to generate the same series again for testing purposes, by entering the same seed.

If *nexpr* is zero, the number returned is the same as the previous one.

If *nexpr* is positive, a new random number is generated.

The random number returned is in the range 0 to 1. Technically, it is never zero and never unity, but all values between 0 and one will be randomly generated and the distribution of numbers will be relatively flat.

Examples

- 1 numfieldc = RND(numfielda)
- 2 numfieldc = RND(2) * 12
- 3 textfieldc = MID\$(x\$, RND(2) * 6 + 1, RND(2) * 12 + 1)
- 4 x#% = RND(y%%)
- 5 ? RND(x%%)

ROW

Purpose

Returns the row position of the insertion point on the screen.

Syntax

ROW (0)

Comments

This function shows how far down the screen the insertion point is. For the column position, see COL.

Examples

- 1 x%% = ROW(0)
- 2 ? ROW(0)

Notes

This function expects you to supply a parameter having a numeric value. Since there is no information for you to give you should always specify zero.

In practice, example 2 would be pointless, because it changes the position of the insertion point in the course of printing it.

See Also

COL, HOME, LOCATE

RUN

Purpose

Executes a program from memory, or loads it from disk and then runs it.

Syntax

RUN [*filename*]

Comments

This will run the program currently in memory when used as a command or as a program statement without the *filename* option. If *filename* is used to specify a program, Superbase loads the program from disk and then runs it.

If *filename* is specified, it must be a string variable or a string constant in quotation marks.

See Also

CHAIN, LOAD, OPEN FORM

SAVE

Purpose

Saves any of the following types of file: program, text, function key, query, and update files.

Syntax

SAVE [TEXT/KEY/QUERY/UPDATE/SET] [*filename*] [,TEXT]

Comments

filename is required with all keywords except SET. Superbase detects files of different types as follows:

- aaa.sbk is a saved function key set
- aaa.sbp is a saved program
- aaa.s bq is a saved query
- aaa.sbt is a saved document (text)
- aaa.sbu is a saved update

If none of the options TEXT, KEY, QUERY, UPDATE or SET is used, Superbase assumes that *filename* refers to an SBP file and attempts to save a program file. If you include TEXT as the last parameter, DML saves a program file as a text file.

Only one of the options, TEXT, KEY, QUERY, UPDATE or SET, can be used at a time.

SAVE SET (with no *filename*) saves the current settings of all the relevant Superbase parameters in file S:SUPERBASE.INI.

See Also

LOAD

SAVE FILE

Purpose

Saves the current file definition.

Syntax

SAVE FILE *sbfname*

Comments

When you create a new file, you can use MAKE to save the file definition. SAVE

FILE, however, must be used after you have edited a file definition with MODIFY.

See Also

MAKE, MODIFY

SAY

Purpose

Converts a text string into speech, using the Amiga's speech synthesis facility.

Syntax

SAY [USING *parameters*] *exprlist*

Comments

The parameters for the USING option are in the order:

Pitch, Inflection, Rate (wpm), Sex, Phonemic.

The following table gives the range of each parameter and its default value:

Parameter	Range	Default	Notes
Pitch	65-320	110	
Inflection	0-1	0	0 is expressive, 1 is monotone
Rate in wpm	40-400	150	
Sex	0-1	0	0 is male, 1 is female
Phonemic	0-1	0	0 translates to phonemes, 1 assumes phonemes

SAY works only with string expressions. To hear an external sound field use the SHOW command.

SD

Purpose

Returns the standard deviation for a range of values.

Syntax

SD (*fieldname / numarray*)

Comments

This function provides a measure of the spread of a range of numeric values from its mean. It only operates with repeated instances of a numeric field or numeric variable such as occur in reports and transaction forms. It can also be used with numeric arrays that are defined within a program. See COUNT.

Examples

- 1 ? SD (n#%)
- 2 Deviance## = SD Scores.Results

Notes

The parentheses around the function's argument are optional.

See Also

COUNT, DIM, GROUP, MAX, MEAN, MIN, REPORT, SUM, VAR

SECS

Purpose

Returns the number of seconds (as would appear in a 'hh:mm:ss' display) from a time field.

Syntax

SECS (*nexpr*)

Comments

nexpr will usually contain a time in milliseconds (thousandths of a second) from a time field or from the result of a TIMEVAL calculation.

Examples

- 1 scnds%% = SECS(timefield)
- 2 scnds%% = SECS(TIMEVAL("10:22:36.875"))

See Also

HRS, MINS, THOUSECS

SELECT

Purpose

A SELECT command of this type retrieves data, and allows you to specify conditions and sorting order. Superbase recognizes the type of SELECT command by the syntax and the presence or absence of certain keywords. The syntax in this case is:

Syntax

```
SELECT ; / ALL / [output_parameters] /var, ...  
[REPORT parameters]  
[WHERE conditions [ASK] [ADD]]  
[ORDER field1 [ASCENDING/DESCENDING] ... [ASK] [ADD]]  
[TO PRINTER / FILE "filename" / "filename"]  
[END SELECT]
```

Comments

This form of SELECT is a query language command and can be used on its own or with other query language commands to form a query section.

; (Semicolon). It is possible to suppress the output of a multi-line SELECT command by using a semicolon:

```
SELECT ;
```

This command will read all the records in the current file without outputting any data. The feature may be used in conjunction with BEFORE SELECT or AFTER SELECT when special processing is required.

output_parameters can be any of the output format parameters as listed in the section which describes the ? commands. In addition, there are three format parameters which can only be used with the SELECT command: ON *file*, AS *heading* and FIELD. The syntax and function of these parameters are described Chapter 14 Defining and Using Queries, in Volume 1.

fields can be one or more field names from one or multiple files. They can be entered on the same line as SELECT itself (as they would be in the Query Definition requester) or they can extend over multiple lines following the SELECT statement.

Optional clauses are other query language statements formed with the commands REPORT, WHERE, and ORDER. They may be entered on the same line, separated by colons, or on separate lines. If the SELECT statement is being used with other reporting commands such as BEFORE GROUP, the REPORT statement should be omitted.

The keyword ADD may follow the ASK keyword in the WHERE and ORDER clauses of a SELECT statement.

The effect of ADD in a WHERE clause is to hide any existing filter conditions, such as those generated in the Form Designer for a multi-file report program, while allowing the user to specify additional conditions. The additional conditions are appended to the existing conditions with the AND keyword.

The effect of ADD in an ORDER clause is to hide any existing order fields, such as those generated in the Form Designer for a multi-level sorted report program, while allowing the user to specify additional order fields. The additional order fields are appended to the existing order fields with comma separators.

END SELECT is used to indicate the end of a query section when the results are to be displayed on the screen. (If you are outputting query results to another device such as the printer, you must use the TO *device* statement, as described below).

It is not always necessary to include END SELECT, but you must provide Superbase 4 with some indication of where the query section finishes and where the rest of the program starts. Otherwise, the statements following the last line in the query section will be interpreted as part of a multi-line SELECT statement. As an alternative to END SELECT, you could use a blank line.

TO *device* specifies the device to which the query output will be sent. If it is not included output is to the screen.

The device options are:

TO PRINTER

Outputs to the printer.

TO FILE *file*

Creates a new SBF file on disk under the file name specified, using the query output.

TO *file*

Outputs to the ASCII file on disk specified by *file*.

Examples

- 1 SELECT Custno.Customer, Firstname.Customer
 Lastname.Customer, Balance.Accounts
 WHERE Custno.Customer = Custno.Accounts AND
 Balance.Accounts > 500
 ORDER Balance.Accounts
 END SELECT
- 2 OPEN FILE "Address"
 SELECT ALL
 WHERE Country LIKE "USA"
 TO FILE "USAddS"

Notes

Example 1 displays the details of all the customers (in the Customers file) whose balance (in the Accounts file) is greater than \$500. Example 2 creates a new file from the Address file, taking only those records where the country is USA.

See Also

SELECT Commands and Query Language Commands in Chapter 15 DML Overview.

SELECT CASE

Purpose

Allows conditional execution of program statements depending on the value of a test expression.

Syntax

```
SELECT CASE test expr
CASE expr 1 [,...expr n]
  statements
[CASE expr 1 [,...expr n] statements]
[CASE ELSE statements]
END SELECT
```

Comments

If the *test expr* matches an expression in the list associated with a CASE clause, then the *statements* following that CASE clause are executed up to the next CASE clause or, for the last CASE clause, up to END SELECT. Control then passes to the statement following END SELECT.

CASE resembles the IF THEN ELSE statement in that it allows program flow to be directed according to the value of an expression, the *test expr* of the SELECT CASE line. However, whereas IF THEN ELSE tests only for true or false, CASE allows you to anticipate a range of values for *test expr* and specify different actions for each outcome. So each CASE clause – a CASE line and its attached expressions and statements – may be compared to an IF THEN statement.

The CASE ELSE *statements* are executed only if the *test expr* does not match any *expr* in a CASE clause. The CASE ELSE clause is useful for handling unforeseen values in *test expr*.

The CASE expression list may include three forms:

```
expr [, expr ...]
expr TO expr
IS relational operator expr
```

See Chapter 15 DML Overview for an explanation of relational operators. The expression list may include all three forms.

Examples

- 1 SELECT CASE ASC(test\$)
 CASE 44
 ? "this is a comma"
 CASE 43, 45
 ? "this is a plus sign or a minus sign"
 CASE 46
 ? "this is a decimal point"
 CASE ELSE
 ? "this is not a comma, a plus, a minus, or a decimal point"
 END SELECT
- 2 SELECT CASE test%%
 CASE 2 TO 4, 8 TO 10, 17, 18, 19, IS > x%%
 ? &5.0 test%%; " matches!"
 END SELECT
- 3 SELECT CASE test\$
 CASE "A", "B", "C", "E" TO "K", IS <> y\$
 ? test\$; " matches!"
 END SELECT

Notes

If you use the TO keyword to indicate a range of values, the smaller value must appear first. Note that -4 is smaller than -2.

If there is no CASE ELSE clause, and the *test expr* is not matched by any *expr*, program execution continues without generating an error.

If an *expr* occurs in more than one CASE clause, only the *statements* associated with the first occurrence are executed.

SELECT CASE statements may be nested; each must have its own END SELECT statement.

See Also

IF ... THEN ... ELSE, GOSUB

SELECT CURRENT

Purpose

Selects the current record.

Syntax

SELECT [FORM] CURRENT [FILE *sbfname*] [INDEX *index*]

Comment

This command has the same effect as the Current Record button on the browsing controls at the bottom of the screen. Use INDEX to select the current record using a different index.

FORM activates the form's link structure.

See Also

Record Selection by SELECT Commands in Chapter 15.

SELECT DUPLICATE

Purpose

Selects the next record with the same key.

Syntax

SELECT [FORM] [PREVIOUS] DUPLICATE [INDEX *index*]

Comments

This command finds the next record with the same key as the current key. The key is the field on which the file is currently indexed. The PREVIOUS option retrieves the previous matching record in the file. If SELECT DUPLICATE fails to find a record with the same key, the current record remains the same and the EOF function is set to true.

FORM can be used when a form has been opened to activate the form's link structure.

Examples

```
1  SELECT FIRST : VIEW: x%% = 1
   lab1: SELECT DUPLICATE
   IF NOT EOF("") THEN VIEW: x%% = x%% + 1: GOTO lab1
   IF x%% > 1 THEN
       ? x%%; " duplicates - strike a key": WAIT a$: CLS
   ENDIF
   SELECT NEXT: IF EOF("") THEN END
   CLS : x%% = 1: VIEW: GOTO lab1
```

Example 1 displays duplicate entries on an index. In Table View, it shows a set of records at a time.

See Also

Record Selection by SELECT Commands in Chapter 15.

SELECT FIRST

Purpose

Selects the first record in the current or specified index sequence.

Syntax

```
SELECT [FORM] FIRST [FILE sbfname] [INDEX index]
```

Comments

This command has the same effect as the First Record button on the browsing controls. Use the INDEX parameter to alter the current index.

FORM can be used when a form has been opened to activate the form's link structure.

Examples

```
1  SELECT FIRST
2  SELECT FIRST FILE "Stock" INDEX Prodcod
```

Notes

Example 1 selects the first record in the current file according to the current index. Example 2 selects the first record in the Stock file according to the Prodcod index.

See Also

Record Selection by SELECT Commands in Chapter 15.

SELECT FORM NEXT/PREVIOUS PAGE

Purpose

Reads and displays the next or previous set of transaction records on a one-to-many relational form.

Syntax

SELECT FORM NEXT/PREVIOUS PAGE

Comments

Where a multi-file application has a one-to-many relationship between two files, you may need to keep more 'many' records on file for each 'one' than can be shown at one time on a form. This command allows you to scroll through the many, or transaction, records page by page, either forwards or backwards.

The RECCOUNT function should be used to test whether any more transaction records remain to be retrieved. RECCOUNT returns the number of transaction records retrieved with the last SELECT FORM command, such as SELECT FORM FIRST, as well as the number retrieved with SELECT FORM NEXT/PREVIOUS PAGE.

Examples

```
1  IF RECCOUNT("**") <> 0 THEN
    SELECT FORM NEXT PAGE
  ELSE
    SELECT FORM NEXT
  ENDIF
```

Notes

Example 1 steps through a master file retrieving all transactions page by page before moving to the next record.

See Also

Record Selection by SELECT Commands in Chapter 15.

SELECT FORM ROW

Purpose

Selects the current transaction row.

Syntax

SELECT FORM ROW *nexpr*

Comments

This command is used with transaction forms to select one from a number of transaction lines. Transaction data on the form is held independently of record data. Consequently, it is necessary to select the transaction you wish to deal with in order to ensure that the record data reflects the transaction data.

Once you have identified a transaction with SELECT FORM ROW, any reference to a field involved in transactions will refer to the instance of that field in that transaction. See UPDATE FORM, ENTER, FORM.

Note that in selecting a transaction you also select the record which holds the data for the fields in the transaction; in other words, you make it the current record.

Examples

```
1  x%% = COUNT quantity.invline
   FOR n%% = 1 TO x%%
   SELECT FORM ROW n%%
   ? quantity.invline
   NEXT
```

Notes

The example displays all the quantity entries in the current form.

See Also

UPDATE FORM ROW, and Record Selection by SELECT Commands in Chapter 15.

SELECT KEY

Purpose

Selects the first record with matching string or numeric value.

Syntax

```
SELECT [FORM] KEY string/nexpr [FILE sbfname] [INDEX index]
```

Comments

This command has the same effect as the Key Lookup button on the browsing controls. It searches the file for the first record whose index field matches the string or numeric value specified. The key expression must be of the same type as the index field; i.e., you should supply a string if the index field is a string field, a numeric value if it is a numeric field.

Unlike the other SELECT commands, this command does not affect the EOF function, but instead sets the FOUND Function (see the example given for FOUND).

FORM can be used when a form has been opened to activate the form's link structure.

Examples

- 1 SELECT KEY "Zollinger"
VIEW
- 2 SELECT KEY "Johnson" FILE "Customer" INDEX Lastname
IF FOUND("Customer") THEN FILE "Customer": VIEW
- 3 SELECT KEY 375 INDEX Number: VIEW

Notes

Example 1 selects the record in the current file whose Lastname field contains the name Zollinger. It assumes that the current file is indexed on Lastname. The program in example 2 selects the record in the Customer file whose Lastname field contains the name Johnson. The Customer file must have already been opened, but it does not have to be the current file. If the program finds a record with a matching key, it uses the FILE command to make the Customer file current and then displays the record.

See Also

Record Selection by SELECT Commands in Chapter 15.

SELECT LAST

Purpose

Selects the last record in the current or specified index sequence.

Syntax

SELECT [FORM] LAST [FILE *sbfname*] [INDEX *index*]

Comments

Has the same effect as the Last Record button on the browsing controls.

FORM can be used when a form has been opened to activate the form's link structure.

See Also

Record Selection by SELECT Commands in Chapter 15.

SELECT NEXT

Purpose

Selects the next record in the current or specified index sequence.

Syntax

SELECT [FORM] NEXT [FILE *sbfname*] [INDEX *index*]

Comment

Has the same effect as the Next Record button on the browsing controls.

FORM can be used when a form has been opened to activate the form's link structure.

See Also

Record Selection by SELECT Commands in Chapter 15.

SELECT PREVIOUS

Purpose

Selects the previous record in the current or specified index sequence.

Syntax

SELECT [FORM] PREVIOUS [FILE *sbfname*] [INDEX *index*]

Comments

This command has the same effect as the Previous button on the browsing controls.

FORM can be used when a form has been opened to activate the form's link structure.

See Also

Record Selection by SELECT Commands in Chapter 15.

SELECT REMOVE

Purpose

Removes the current record in the current file or another open file.

Syntax

SELECT [FORM] REMOVE [FILE *sbfname*]

Comments

This command has the same effect as the Cut command on the Edit menu, except that no data is placed on the clipboard.

FORM can be used when a form has been opened to activate the form's link structure.

See Also

REMOVE, REMOVE FROM

SELECT WHERE

Purpose

Sets the record selection filter.

Syntax

SELECT [FORM] WHERE [[FILE *sbfname*] [*conditions*]/[ASK]]

Comments

Use this command to define the filter conditions or to remove the filter. It has the same effect as the Filter button on the browsing controls, except that it does not select a record. If you wish to select the first record that matches the filter conditions, you need to execute two commands: SELECT WHERE to define the filter, followed by SELECT FIRST to select the record.

The FORM parameter is used to activate the form's link structure.

conditions is set up in the same way as the string gadget in the Filter requester. If not specified, the current filter conditions are cleared.

SELECT WHERE can only be used to set a single file filter. If you enter the name of a field which also occurs in another open file, you should include the file name as an extension. Otherwise, Superbase may assume you are attempting to use this command in a multi- file operation, and will issue the error message:

This Where statement must be single file

If you wish to set a multi-file filter – to select records whose field data matches the data in another file – use the LOOKUP function or the query language command WHERE.

With commands that set a single file filter using WHERE (for IMPORT, EXPORT as well as SELECT WHERE) you can include ASK as the parameter to the WHERE statement. The effect of ASK is to display the standard filter requester when the command is executed, thus allowing the user to set a filter while a program is running. If a filter is present when ASK is executed, the filter expressions will be presented as a default for the requester.

Examples are:

SELECT WHERE ASK

In this example, ASK causes the filter requester to be presented. Entering the following filter line (at run-time):

Lastname LIKE "[r-z]*"

would have the same effect as executing the command:

SELECT WHERE Lastname LIKE "[r-z]*"

However, a partial filter like this is not allowed:

```
SELECT WHERE datefield ASK
```

As with the other Record SELECT commands, the FORM parameter is used to activate the form's link structure.

If the user clicks the Cancel button on a SELECT WHERE ASK requester, a CTRL+C error is generated (error 11). This should be trapped and handled with the ON ERROR, ERRNO, ERR\$, and RESUME group of commands.

The keyword ADD may follow the ASK keyword in the WHERE clause of a SELECT statement. The effect of ADD in a WHERE clause is to hide any existing filter conditions, such as those generated in the Form Designer for a multi-file report program, while allowing the user to specify additional conditions. The additional conditions are appended to the existing conditions with the AND keyword.

Examples

- 1 SELECT WHERE fielda LIKE "[a-c]"
SELECT FIRST
WHILE NOT EOF("")
SELECT NEXT
WEND
- 2 SELECT WHERE FILE "Stock" fieldb LIKE "[a-c]"
- 3 SELECT WHERE
- 4 SELECT FORM WHERE Date_of_birth > "1 Jan 1950"
- 5 SELECT WHERE ASK

Notes

Once set, the browsing controls filter remains active until it is cleared. Example 3 clears the filter which may have been set by a previous SELECT WHERE command or by direct entry in the filter requester.

See Also

Query Specification by SELECT Commands, and Query Language Commands, in Chapter 15.

SER

Purpose

Returns the total number of records that have been created in a file.

Syntax

SER (*filename*)

Comments

You can use the SER function to assign a serial number to each record in a file. To do this, you need to define a field which will hold the serial number. It should be defined as a constant field and should have SER("filename") as its constant formula. When you create the first record, it will be given the value 1. This value will then be incremented by one for each record you add to the file.

The difference between SER and RECCOUNT is that SER gives the total number of records that have been created, irrespective of whether they have been deleted later. RECCOUNT returns the number of records currently in the file.

Examples

```
1  ? SER("Stock")

2  BLANK
   Recno.Stock = SER("STOCK")
   Price.Stock = 14.95
   Description.Stock = "Widget"
   STORE
```

Notes

If the file is explicitly named within quotation marks, you will have to change it in the file definition following a REORGANIZE command or a Query to File where the new file has a different name. The new name should replace the old name.

See Also

RECCOUNT

SET

Purpose

Reads a text file and executes any commands in the file, or assigns variables from a text file.

Syntax

SET *filename*

Comments

This command reads in a text file and executes it as if it were a sequence of commands. The file, therefore, must contain valid DML commands.

If the file holds a set of variables – which have previously been saved to disk by ? MEMORY – the variable assignments are executed. This provides a way of transferring variables between different programs when CHAIN is not appropriate.

For example, Program 'a' can set up variables for Program 'c,' but Program 'a' CHAINs to Program 'b.' Another application would be to communicate variables between different programs which are run on different days.

SET must be placed last on a program line.

Examples

```
1  .....
   process a
   .....
   OPEN "aaa" FOR OUTPUT: ? MEMORY: CLOSE OUTPUT: DISPLAY
   another program
   SET "aaa"
   .....
   process b
2  SET "abc"
```

'abc' could be an ASCII file which contains a set of function key assignments.

See Also

EXECUTE, LET

SET BUFFERS

Purpose

Sets the number of buffers Superbase uses as a disk cache.

Syntax

SET BUFFERS *expr*

Comments

Operates in the same way as the Buffers option in Set System Options, and allocates 512 bytes memory space for each buffer.

expr can have a value from 1 to 99.

SET BUFFERS only sets the size of the disk buffers for the current session. Use the command SAVE SET to store the value in the S:SUPERBASE.INI file.

Examples

```
1 SET BUFFERS 24
```

SET EDIT

Purpose

Sets auto-entry to the Program Editor on or off.

Syntax

SET EDIT ON / OFF

Comments

Normally when a program is interrupted – after CTRL+C has been pressed or the Stop button has been clicked, or when an error has occurred – Superbase opens the Program Editor window and makes the cursor active. SET EDIT OFF switches this feature off; if the window has been closed beforehand, it will remain closed when the program is stopped.

See Also

BREAK

SET EDIT ATTR

Purpose

Determines the color settings of field data during data entry to provide an editing focus for the user where the presence of colored fields on a form could be confusing.

Syntax

SET EDIT ATTR [ON/OFF]

Comments

When you enter data in a form whose fields have been given color attributes (e.g. white text on a red background), the data in a field will be shown with those color attributes as it is typed in, or as you enter the field for editing; all blank characters after the end of the data will be in monochrome. For uniformity, it may be preferable to have all the data appear in monochrome (i.e. black text on a white background). This has the effect of "highlighting" the field which is being edited.

The SET EDIT ATTR command may be executed either before an ENTER command or before native mode data entry.

Examples

1 SET EDIT ATTR ON

Sets the field to monochrome during data entry or editing.

2 SET EDIT ATTR OFF

Sets the field to the colors defined in the Form Designer.

SET FIELDS

Purpose

Determines whether fieldnames are shown in default views.

Syntax

SET FIELDS ON/OFF

Comments

This command is equivalent to the Set Show Field Names menu command. The default condition is SET FIELDS ON.

SET FILE MAX

Purpose

Sets Extended File handling on or off.

Syntax

SET [FILE] MAX ON/OFF

Comments

Superbase supports up to 1 billion records per file, but only if you issue a SET FILE MAX ON command.

By default this is OFF, which makes Superbase compatible with earlier releases of compatible products. The maximum file size is then 16 million records or 1073 Megabytes per file, whichever is the smaller (which is more than sufficient for most applications).

Examples

```
1 SET FILE MAX ON
```

This command ensures that the system works properly with files of more than 16 million records or occupying more than 1073 Megabytes, i.e. Extended files.

See Also

CREATE

SET FN KEY

Purpose

Allows detection of function key press within a DML program.

Syntax

SET FN KEY ON / OFF

Comments

By default, the WAIT KEY or GET commands detect a function key press and take the first character of the string that has been previously assigned to the function key. SET FN KEY OFF can be used to disable function key detection. It also prevents functions key strings which have '!' as their first character from being treated as commands: if FN KEY has been set to OFF, the function string

will not be executed when the key is pressed, but will be treated as text.

Examples

```
1 SET FN KEY ON
  WAIT KEY: GET a$
  IF a$ <> "y" THEN QUIT
```

SET FORM

See SET View or Form

SET FORM ATTR

Purpose

Sets the attributes for form objects.

Syntax

SET FORM ATTR *nxpr1* [,*nxpr2* [,*nxpr3* [,*nxpr4*]]]

Comments

Nxpr1 defines the drawing modes for the object. Each mode has a specific numeric value. If multiple attributes are required for an object, simply add the values together.

Value	Mode	Effect
1	Pen and paper	Uses both pen and paper
2	Rounded	Areas and boxes have rounded corners
4	Border	Areas and fields have borders
16	Right align	Fields and variables are right aligned
32	Center	Fields and variables are centered
64	Scaled	Images are scaled to fit box

Nxpr2 defines further attributes for objects. As with drawing modes, multiple attributes may be set by adding values together.

Value	Attribute	Effect
-------	-----------	--------

1	Currency	Format variable data to 2 decimal places
2	Read only	Field or variable cannot be modified
4	Skip entry	Field or variable skipped during data entry
8	Non printing	Object will never be printed
16	Pushbutton	Command rendered as a pushbutton

Nexpr3 defines the line type. The range of values is 1 through 6.

Nexpr4 defines the pattern type. The range of values is 1 through 15.

The value 0 may be used in any of the first three expressions to leave the current setting unaffected.

Example

SET FORM ATTR 7,0,2

Defines the drawing mode for pen and paper, rounded corners, and borders; leaves the attributes unchanged; sets the line type to 2.

See Also

ADD FORM, CREATE FORM, and the section Form Creation Using DML in Chapter 15.

SET FORM BF

Purpose

Sets or clears the boldface attribute for text or fields.

Syntax

SET FORM BF [ON/OFF]

Comments

The command causes all text and field objects added to the form to be shown in bold until SET FORM BF OFF is executed.

See Also

ADD FORM, CREATE FORM, and the section Form Creation Using DML in Chapter 15.

SET FORM BG

Purpose

Sets the background color for form objects.

Syntax

SET FORM BG *nexpr*

Comments

Nexpr defines the paper color that is used as background when adding form objects. The background color only appears if the object is drawn while the 'pen and paper' attribute is set (see SET FORM ATTR, below). The color appears in areas drawn with a non-solid pattern, text, fields, checkboxes, radio buttons, calculations, commands, and pushbuttons.

Color values are as for SET FORM FG, above.

See Also

ADD FORM, CREATE FORM, and the section Form Creation Using DML in Chapter 15.

SET FORM FG

Purpose

Sets the foreground and outline colors for form objects.

Syntax

SET FORM FG *nexpr1*[,*nexpr2*]

Comments

SET FORM FG sets the foreground and outline colors that are used when adding new form objects. *Nexpr1* denotes the foreground color, which is the pen color used for all text, lines, boxes, areas and patterns. *Nexpr2* denotes the outline color, which appears in bordered fields, areas, and pushbuttons.

Values from 0 to 255 may be valid. The default colors 0 to 15 are as follows:

0 White	8 Light Grey
1 Black	9 Dark Grey
2 Red	10 Dark Red
3 Green	11 Dark Green
4 Blue	12 Dark Blue
5 Cyan	13 Dark Cyan
6 Yellow	14 Dark Yellow
7 Magenta	15 Dark Magenta

Values greater than 15 refer to a range of other colors dependent on your configuration.

See Also

ADD FORM, CREATE FORM, and the section Form Creation Using DML in Chapter 15.

SET FORM IT

Purpose

Sets or clears the italic attribute for text or fields.

Syntax

SET FORM IT [ON/OFF]

Comments

The command causes all text and field objects added to the form to be italicized until SET FORM IT OFF is executed.

See Also

ADD FORM, CREATE FORM, and the section Form Creation Using DML in Chapter 15.

SET FORM PAGE

Purpose

Sets the form page.

Syntax

SET FORM PAGE *nexpr*

Comments

SET FORM PAGE sets the page to which new form objects will be added. The page must already exist.

See Also

ADD FORM, CREATE FORM, and the section Form Creation Using DML in Chapter 15.

SET FORM TEXT

Purpose

Sets the font for text and field objects.

Syntax

SET FORM TEXT *stexpr*

Comments

SET FORM TEXT defines the font to be used when adding text or field objects. *Stexpr* contains font name and point size.

Example

SET FORM TEXT "Courier,11,1"

Defines the font as eleven-point Courier printer font.

See Also

ADD FORM, CREATE FORM, and the section Form Creation Using DML in Chapter 15.

SET FORM UL

Purpose

Sets or clears the underline attribute for text or fields.

Syntax

SET FORM UL [ON/OFF]

Comments

The command causes all text and field objects added to the form to be underlined until SET FORM UL OFF is executed.

See Also

ADD FORM, CREATE FORM, and the section Form Creation Using DML in Chapter 15.

SET FORM USING

Purpose

Changes the units of measurement used in the DML form creation commands.

Syntax

SET FORM USING *nexpr*

Comments

This command allows for the units used in the CREATE FORM, CREATE PAGE, and ADD FORM to be changed. The default units are millimetres. The options are as follows:

0	Millimetres
1	Hundredths of an inch
2	Pixels (screen pixels)

See Also

ADD FORM, CREATE FORM, and the section Form Creation Using DML in Chapter 15.

SET HEADING

Purpose

Sets the title for the Superbase window.

Syntax

SET HEADING *strexpr*

Comments

The title bar for the Superbase window normally shows the name of the current open file. SET HEADING allows you to replace this with your own heading. To restore the original heading, use SET HEADING "".

Examples

- 1 SET HEADING "Invoice Form"
- 2 SET HEADING ""

See Also

HEADING, SET REQUEST, SET STATUS

SET NOW

Purpose

Sets the time held in the system variable NOW.

Syntax

SET NOW *strexpr*

Comments

strexpr must contain a time in the current time format as set with Date Format in the Set menu or with the DATABASE command.

Examples

- 1 SET NOW "14:35:08"
- 2 DATABASE "hh:mm:ss.s"
SET NOW "23:59:59.999"

See Also

NOW, TODAY

SET PAGE

See SET View or Form

SET PAGING

Purpose

Sets paging on or off.

Syntax

SET PAGING [ON/OFF]

Comments

When used without ON/OFF, SET PAGING acts as toggle and operates in the same way as the SET PAGING menu option. With ON or OFF, it sets paging accordingly.

Note that the commands EJECT, LOCATE, and FOOTING do not work when the current output device is the screen and PAGING is off. You can, however, use LOCATE with paging off to set the column position – but not the row position.

Examples

```
1 SET PAGING OFF
```

See Also

?, HOME, LOCATE, SELECT

SET PG

Purpose

Sets the size of the printable area on printed pages.

Syntax

[SET] PG [*rows* [, *columns* [, *ss*]]] / [*strexpr*]

Comments

The two forms of this command apply to the Amiga ‘preferences’ printer and to the printer under Superbase control.

With the 'preferences' printer, use the form:

SET PG *rows* [,*columns* [,*ss*]]

where *rows* specifies the maximum number of lines to be printed on a page, after which Superbase sends a form feed instruction to the printer, *columns* specifies the maximum number of characters per line, and *ss* stands for single sheet and takes a value of 0 to specify continuous (fan-fold) paper, or 1 to specify single sheet paper. SET is optional.

With a Superbase-controlled printer, use the form:

SET PG *strexpr*

where *strexpr* is a string of four decimal numbers, separated by commas, giving the required dimensions of the left, right, top and bottom margins.

Examples

1 SET PG 50, 65, 0

Tells Superbase to print no more than 50 lines per page and 65 characters per line on the preferences printer, and that continuous stationery is in use.

2 SET PG"2.50, 2.50, 3.85, 3.85"

Tells Superbase to allow 2.50cm margins at each side, and 3.85cm margins at the top and bottom, of pages printed on the Superbase printer. Use cm or inches according to the current units setting in the Page Setup requester.

See Also

SET PRINTER REQUEST

SET PG REQUEST

Purpose

Allows access to the Page Setup requester used by the Set Printer Setup menu command.

Syntax

SET PG REQUEST

Comments

This command causes Superbase to show the font and size combinations available for the currently selected printer. The user's choice becomes the current font.

SET POSITION

Purpose

Sets the position and size of windows.

Syntax

SET POSITION [FOR *windowname*] *nxpr1,nxpr2,nxpr3,nxpr4*

Comments

windowname, if specified, must be one of: "main", "controls" or "editor". If *windowname* is not specified, then "main" is assumed.

nxpr1 and *nxpr2* set the x and y co-ordinates of the window. *nxpr3* sets the width of the window, and *nxpr4* sets the height.

All values are in screen co-ordinates.

If SET POSITION is used without arguments, it locks the size and position of the specified window.

Examples

```
1 SET POSITION FOR "EDITOR" x%%, y%%, w%%, h%%
```

Sets the position and size of the window used by the Text Editor and the DML Program Editor.

SET PRINTER

Purpose

Loads the printer control file for the specified printer.

Syntax

SET PRINTER *strexpr*

Comments

This is the program equivalent of selecting a printer from the first requester displayed when you select the Printer Setup command from the Project menu.

Example

```
1 SET PRINTER "Epson"
```

Loads printer control file SB_EPSON.INI and makes Epson the current printer.

SET PRINTER REQUEST

Purpose

Invokes the Printer Setup requester.

Syntax

SET PRINTER REQUEST

Comments

Use this command to select a printer from those available on your computer, and/or to change page size, margin settings, typeface and size, and so on.

SET RECORD

See SET View or Form

SET REQUEST

Purpose

Specifies confirmation messages.

Syntax

SET REQUEST [ON/OFF] / [*nexpr* [OR *nexpr*]]

Comments

The SET REQUEST command allows users to define which of the Superbase confirmation messages, such as 'Record has been modified', are switched off.

SET REQUEST OFF switches off the three messages that request confirmation of saving of record, form, or file definition data following modification. SET REQUEST ON switches the three messages back on.

The *nexpr* alternative allows greater precision in switching messages on and off. *nexpr* is either a specific number or an OR expression that evaluates to a number. The basic values are:

0	All messages on
1	Record or Form message off
2	File message off
4	Text message off
8	Program message off
16	End of file message off
32	Stop OK message off
64	Save this record/form? request off

128	Continue data entry? request off
255	All messages off

You can specify *nexpr* by either adding or OR-ing the above values. For example:

```
SET REQUEST 8 + 16
SET REQUEST 24
SET REQUEST 8 OR 16
```

all specify both Program and End of file messages are to be turned off.

SET REQUEST HEADING

Purpose

Allows Superbase system requesters to be displayed with a custom title.

Syntax

SET REQUEST HEADING *stexpr*

Comments

This command affects all information and error messages, the printing process requester, the communications process requester, and REQUEST requesters with a type number above 99.

The *stexpr* also appears in the title bars of the main, text editor and program editor windows.

Examples

```
SET REQUEST HEADING "Garagebase"
REQUEST "Exit Application?", "", 129, a%
```

Sets the requester title bar so that the user does not realize that the application is developed with Superbase.

SET STATUS

Purpose

This command is included for compatibility only. It may be used in DML programs written for other computers, but on the Amiga it has no effect.

SET TABLE

See SET View or Form

SET TEXT

Purpose

Select a font for printing.

Syntax

SET TEXT *strexpr*

Comments

This command allows you to define a font. The *strexpr* follows the form of a Font line in the current printer control file:

Examples

```
1 SET TEXT "Line Printer 8.5"
```

Sets the font to Line Printer, 8.5-point, assuming that such a combination is available on the current printer.

Caution Superbase uses the font size of the current line when deciding whether to print a footing. If the font size of the footing is significantly larger or smaller, formatting errors may occur.

SET TODAY

Purpose

Sets the date held in the system variable TODAY.

Syntax

SET TODAY *strexpr*

Comments

strexpr must contain a date in the current date format as set with DATABASE or with Date Format in the Set menu.

Examples

- 1 SET TODAY "15 May 1988"
- 2 DATABASE "mm-dd-yy"
SET TODAY "12-31-87"

See Also

NOW, TODAY

SET View or Form

Purpose

Sets the default view or displays a form.

Syntax

SET TABLE / PAGE / RECORD / FORM

Comments

This command works in the same way as the equivalent Set menu commands, except that it does not automatically display a record (use VIEW).

Caution In Superbase Professional version 3.02 and earlier, the comand SET FORM does not display a form, but shows a record in what is now known as PAGE view. DML programs that use SET FORM in the earlier way must be edited to reflect the new syntax.

Examples

- 1 SET FORM
- 2 SET TABLE : VIEW

See Also

CLOSE FORM, OPEN FORM, VIEW

SGN

Purpose

Finds the sign of a number.

Syntax

SGN (*nexpr*)

Comments

This function returns the positive value of 1 if *nexpr* is positive and returns the negative value -1 if *nexpr* is negative.

Examples

```
1  numfieldc = SGN(numfielda)
2  IF SGN(datefielda - datefieldb) THEN GOTO lab1
3  x%% = SGN(y%%)
4  x%% = SGN(y%% * numfielda * (datefielda - datefieldb))
5  x%% = SGN(VAL(RIGHT$(textfielda, 5)))
6  ? SGN(x%%)
```

Notes

Example 2 tests whether **datefieldb** is later than **datefielda**.

See Also

ABS

SHOW

Purpose

Shows an external file.

Syntax

SHOW [*field* [,M]][S][A]] / [*strexpr*]

Comments

SHOW is the program equivalent of the camera button at the bottom of the screen.

It displays a picture or text from an external file. *field* must be a field which holds the name of the external file, but it does not have to be a field in the current file: if you add a file name extension to the field name, you can display pictures using other open database files.

As an alternative to specifying an external file field, *strexpr* allows you to specify the name of an external file.

The M and S parameters are appended to the name of the external image file to be shown, for example:

SHOW "eagle.pcx,ms"

M causes an image to be loaded and displayed in dithered monochrome form; this allows color images to be displayed using much less memory.

S causes an image to be scaled down to fit an external field box on a form *as it is read in*; again, this reduces memory requirements for image display. This feature is the same as the Scale feature in the Form Designer.

Parameter A causes aspect ratio to be preserved. Aspect ratio refers to the fact that the vertical and horizontal dimensions of any given image have only one correct relationship. If an image is distorted either vertically or horizontally, its aspect ratio is lost. In some applications this may not matter, but in others the aspect ratio must be preserved.

If the S and the A parameters are used together, the A parameter takes priority. Also, the Form Designer's Image Scale attribute is overridden by the A parameter.

- Note that the parameters must form part of the filename literal – they fall within the quotation marks.

When *field* or *strexpr* is not given, SHOW removes the picture from the screen.

As well as graphical images, SHOW can also replay files of digitized sound samples. See Chapter 28 of Volume 1.

See Also

FORM SHOW

SIN

Purpose

Returns the sine of an angle measured in radians.

Syntax

SIN (*nexpr*)

Comments

This function returns the sine of the angle in *nexpr*, where the angle is measured in radians. To convert degrees to radians, multiply by $\text{PI}/180$.

Examples

- 1 numfieldc = SIN(numfielda)
- 2 x#% = SIN(y#%)
- 3 x#% = SIN(VAL(x\$))
- 4 x#% = SIN(ndegs#% * PI / 180)

See Also

ATN, COS, TAN

SPACE\$

Purpose

Returns a string containing the specified number of spaces.

Syntax

SPACE\$ (*nexpr*)

Comments

The result of *nexpr* must be value in the range 0 to 255.

Examples

- 1 textfieldc = SPACE\$(6) + "Indented text"
- 2 x\$ = SPACE\$(40 - LEN(y\$))

Notes

If you specify too many spaces for a text field, the field is just filled.

See Also

PAD\$, REPLICATE

SQR

Purpose

Returns the square root of a number.

Syntax

SQR (*nexpr*)

Comments

The function returns the square root of the number specified by *nexpr*. If *nexpr* is less than zero, the function returns the error message 'invalid number.'

Examples

- 1 numfieldc = SQR(numfielda)
- 2 numfieldc = SQR(2 * numfielda)
- 3 textfieldc = STR\$(SQR(VAL(x\$)))
- 4 x#% = SQR(y#%)

STORE

Purpose

Stores the current record.

Syntax

STORE [,0/1/2] [FORM]/[FILE *sbfname*]

Comments

This command stores the record in memory for the current file. If a filename is specified with *sbfname*, the current record for that file is stored.

STORE is equivalent to the Save option on the Record menu.

If the FORM parameter is used all the records in the current form are saved; otherwise STORE only saves the record that belongs to the current file.

The FILE parameter allows you to save a record in a file which is open but not current.

STORE also allows you to store records faster than usual, with the drawback that the data is not secured automatically. This means that you can enter a batch of

records and wait until you are ready to secure them, thus saving time during data entry.

STORE,1 stores the current record without updating the file header. The time taken to save the data on disk will be reduced, but the data is not secure. A power loss, for example, would result in the entered data being lost. Note that the command only operates on the current record, i.e. it does not set this mode for future STORE commands.

STORE,2 makes the file secure. It does not save the current record as such, but updates the control information in the file header. *You must execute STORE, 2 after STORE, 1 in order to secure the data.*

STORE,2 does NOT take the FORM or FILE parameters. It secures all unsecured data automatically.

STORE,0 is optional and therefore is the same as STORE on its own. Data is secured as it is written directly to disk.

Examples

```
1  BLANK
   Firstname = "John"
   Lastname = "Roberts"
   Street = "15 Richmond Way"
   .....
   .....
   STORE
```

See Also

BLANK, ENTER

STR\$

Purpose

Returns the text equivalent of a numeric expression.

Syntax

STR\$ (*nexpr* [, *nexpri* [, *nexprd*]] / [, *numeric-format-string*])

Comments

STR\$ converts data which is held in a numeric variable or numeric field into a text string.

nexpri specifies the number of integers before the decimal point and should be set large enough to avoid overflow display. *nexprd* specifies the number of digits after the decimal point. The maximum numeric format in Superbase is a total of 14 integers, so *nexpri* plus *nexprd* must be less than 15.

As an alternative to using *nexpri* and *nexprd*, you can specify the numeric format as a string (see NUMBASE). If these parameters are not used, the default numeric format set by Number Format on the Set menu or by the most recent use of NUMBASE will be taken.

The complementary function to STR\$ is VAL.

Examples

- 1 textfieldc = STR\$(numfielda)
- 2 textfieldc = STR\$(numfielda, 5, 0)
- 3 x\$ = STR\$(165.4444, "z999999.00")
- 4 x#% = VAL(STR\$(6.66666, "9.999"))

Notes

Example 4 assigns a value of 6.667 to x%; it demonstrates the use of the STR\$ and VAL functions to round a number. See also FIX.

See Also

? &, NUMBASE, VAL

SUM

Purpose

Returns the total for a range of values.

Syntax

SUM (*fieldname / numarray*)

Comments

This function can only be used with numeric fields in reports and transaction forms, and with numeric arrays defined in programs or transactions. See COUNT.

Examples

- 1 ? SUM (Line_Value)
- 2 IF SUM tot#% > 250 THEN . . .

Notes

The parentheses are optional.

See Also

COUNT, DIM, GROUP, MAX, MEAN, MIN, REPORT, SD, VAR

TAN

Purpose

Returns the tangent of an angle measured in radians.

Syntax

TAN (*nexpr*)

Comments

The function returns the tangent of the angle in *nexpr* measured in radians.

The complementary function of TAN is ATN.

Examples

- 1 numfieldc = TAN(numfielda)
- 2 x#% = TAN(y#%)
- 3 x#% = TAN(VAL(x\$))
- 4 ? TAN(ndegs#% * PI / 180)

See Also

ATN, COS, SIN

THOUSECS

Purpose

Takes a numeric value and returns the number of thousandths of a second left over after subtracting the number of seconds.

Syntax

THOUSECS (*nexpr*)

Comments

nexpr will usually contain a time in milliseconds from a time field or the result of a TIMEVAL calculation. THOUSECS returns the same result as *nexpr* MOD 1000.

- 1 x%% = THOUSECS(timefield)

See Also

HRS, MINS, SECS

TIMES

Purpose

Returns the time in string format from a numeric value which gives the time in thousandths of a second.

Syntax

TIMES (*nexp* [,*timeformat*])

Comments

The second argument for this function, *timeformat*, allows you to specify the format the time string will have. It must conform to the rules for Superbase time formats given in the keyword entry for DATABASE.

Examples

- 1 x\$ = TIMES(timefield)
- 2 ? TIMES(NOW, "hh:mm:ss am")

See Also

DATABASE, TIMEVAL

TIMEVAL

Purpose

Returns the value of a time string in thousandths of a second.

Syntax

TIMEVAL (*strexpr*)

Comments

strexpr must contain the time in a valid time format. See the keyword entry for DATABASE for a list of acceptable time formats.

Examples

- 1 t&% = TIMEVAL("10:22am")
- 2 t&% = TIMEVAL("14:03:12.201")

See Also

TIMES

TODAY

Purpose

Gives the system date.

Syntax

TODAY

Comments

TODAY shows the date in the date format as set with the Date Format option in the Set menu, or as set by the DATEBASE command. If your computer has a real-time clock or you have set the system date, TODAY gives the current date. Otherwise, it gives the default system date. To set the system date, use SET TODAY.

TODAY holds the date as a julian date number. Superbase automatically translates this into the current date format when you display the date using ? TODAY.

Examples

- 1 ? TODAY
- 2 datefield = TODAY
- 3 ? MONTH(TODAY)

See Also

DATE\$, NOW

TRIM\$

Purpose

Trims trailing spaces from a string or a text field.

Syntax

TRIM\$ (*strexpr*)

Comments

This returns the string consisting of the original string specified by *strexpr* with any trailing spaces eliminated.

Examples

```
1 stringfieldc = TRIM$(textfielda)
2 x$ = TRIM$(textfieldc.filea)
3 ? LEN(x$); LEN(TRIM$(x$))
```

See Also

LTRIM\$

UCASE\$

Purpose

Returns the upper case equivalent of a text string or a text field.

Syntax

UCASE\$ (*strexpr*)

Comments

UCASE\$ returns the upper case equivalent of the lowercase alphabet; no other characters, including those already in upper case, are affected.

The complementary function of UCASE\$ is LCASE\$.

Examples

```
1 textfieldc = UCASE$(textfielda)
2 x$ = UCASE$(y$)
3 x$ = UCASE$("abcdef")
```

Notes

If you wish to set the first letter of a string to upper case, leaving the rest in lower case, you can do so using the FCASE\$ function.

See Also

FCASE\$, LCASE\$

UPDATE

Purpose

Performs a relational update.

Syntax

UPDATE [*calclist*] [WHERE *conditions*] [END UPDATE]

Comments

UPDATE on its own runs the update in memory. This may have been loaded from disk with the LOAD UPDATE command, or it may have been created in the same session using the Process menu option Update Edit.

By specifying *calclist* and *conditions*, you can also use UPDATE to define an update and then run it. *calclist* corresponds to the command line set in the Update Fields requester; *conditions* corresponds to the filter which is set in the Update Filter requester. The first specifies how the records are updated, the second specifies which records are to be updated.

calclist should consist of one or more statements assigning a value to fields in the current file. For example:

Price = Price * 1.05

or

Bank_rate.Deposits = Bank_rate.Bank

The first *calclist* statement must follow UPDATE on the same line; any other statements may be entered on the same line separated by colons, or on separate lines.

WHERE *conditions* may be entered on the same line as UPDATE separated from the last *calclist statement* by a colon, or on a separate line.

These statements form part of an Update program section, headed by the UPDATE program command and ending with END UPDATE.

The END UPDATE command must be included if the Update section is followed by other statements in a program. Otherwise Superbase will regard these as belonging to the Update section. As an alternative to using this command, you can terminate the section with a colon or a blank line.

UPDATE is a multi-file command, so both *conditions* and *calclist* can refer to more than one file. In this case, the first condition in the update filter must establish a join between two files.

Examples

```
1  LOAD UPDATE "Newrate": UPDATE
```

Loads the Update file Newrate from disk, and then runs it.

```
2  UPDATE Price.Orders = Price.Stock
   WHERE Product_Code.Orders = Product_Code.Stock AND
   Order_date > "15 July 1987"
   END UPDATE
```

Updates prices in the Orders file on the basis of the price details in the Stock file.

Notes

calclist should not modify an indexed field that is used by Superbase to optimize the update performance; i.e., the WHERE clause must not contain a relational expression such as 'Name > "Smith".'

If this is unavoidable, disable optimization by placing parentheses around the expression: '(Name > "Smith").'

UPDATE FORM ROW

Purpose

Copies data from the current record or records to a specified transaction line.

Syntax

```
UPDATE FORM ROW nexpr
```

Comments

This command is only used with transaction forms. It updates the fields in the transaction line specified with *nexpr* taking the data from the current record. To redisplay use the FORM command.

See SELECT FORM, ENTER, FORM.

Examples

```
1  BLANK FORM
   SELECT FIRST
   FOR n%% = 1 TO 10
   UPDATE FORM ROW n%%
   SELECT NEXT
   NEXT
   FORM
```


Notes

The example program fills the first ten lines in the transaction form with data from the first ten records in the current file – it assumes that the fields in the file are the same as those on the form.

See Also

SELECT FORM ROW

VAL

Purpose

Returns the numeric value of a text string.

Syntax

VAL (*strexpr*)

Comments

The function returns the numeric value of the number (if any) in the lefthand end of the string or substring specified in *strexpr*. In cases where *strexpr* does not contain a number or where the leftmost character of *strexpr* is not numeric, the function returns 0.

The complementary function of VAL is STR\$.

Examples

- 1 numfieldc = VAL(textfielda)
- 2 numfieldc = VAL(RIGHT\$(textfielda, 8))
- 3 IF VAL(textfielda) > 1 THEN ...
- 4 x#% = VAL("12.45456")
- 5 x#% = VAL(x\$)
- 6 x#% = VAL(STR\$(5.66666, "9.999"))

Notes

Example 6 demonstrates the use the VAL and STR\$ functions to round a number. See also FIX.

See Also

STR\$

VAR

Purpose

Returns the variance from the mean of a range of values.

Syntax

VAR (*fieldname* / *numarray*)

Comments

fieldname must be a numeric field whose data is repeated in a report or a field in a transaction line. *numarray* can be a numeric calculation variable that occurs in a transaction line or a numeric array in a program. The parentheses are optional.

Examples

- 1 IF VAR Score.Results > v#% THEN ...
- 2 x#% = VAR scores%%

See Also

COUNT, DIM, GROUP, MAX, MEAN, MIN, REPORT, SD, SUM

VIEW

Purpose

Displays the current record in the current file.

Syntax

VIEW

Comments

Allows the user to see the current record in the current file in the view format specified by the SET *view mode* command. It can also be used to redisplay the current Form.

See Also

FORM, SET VIEW/FORM

WAIT

Purpose

Waits for a specified time or until a key is pressed or until the mouse button is clicked.

Syntax

WAIT [MOUSE/MENU/KEY/PANEL [LOCK] [OFF]] / [FOR *time/nexpr*]
/ [*var/field*]

Comments

WAIT MOUSE waits for a mouse click in the data window.

WAIT MENU waits for a selection from a user defined menu.

WAIT KEY waits for a key to be typed while the data window is active.

This can be a function key or an ordinary key. If a function key is pressed and it contains a command string prefixed with the '!' character, Superbase executes the command. If the function key does not contain a command, control moves from WAIT KEY to the next statement, which should be GET *stexpr*; the *stexpr* will contain the first letter of the string attached to the function key. See also SET FN KEY ON/OFF.

WAIT PANEL activates the browsing controls at the bottom of the screen and processes user selections until the stop button is selected. The OFF keyword used with PANEL accepts input from the keyboard equivalents to the browsing controls, when the PANEL OFF command has removed the display of the browsing controls.

The optional parameter LOCK may be used with WAIT PANEL. This prevents the user from switching to another page of the form while using the browsing controls under program control.

These options may be combined in any order using the OR keyword. For example:

WAIT MOUSE OR MENU OR KEY

This would suspend Superbase operations until any of the specified events occurred. The user program is then responsible for determining which type of event or events has actually occurred. Note that the OR keyword is optional.

Wait with the FOR parameter waits for a given number of seconds or until a given time.

FOR *nexpr* implies 'wait for *nexpr* seconds.'

FOR *time* implies 'wait until the system clock reaches *time*.' where the time is given as a string expression in the current time format.

var/field implies 'wait until a key is pressed, and then place it in *var* or *field*.' If you follow WAIT with a numeric variable or numeric field, it will only accept a

number. In other words, pressing any key except the keys with the digits 0 to 9, will have no effect.

WAIT *varfield* makes the database window active before it starts waiting for input, so if the window becomes inactive for any reason you cannot reactivate it. If this is a problem for your application, use GET instead.

Note that control characters can be returned in WAIT statements. If these are saved in a file which is subsequently LISTed, they will generate the 'File contains non-text characters' error. Note also that CTRL+C, the standard interrupt key, does not stop program execution if returned in WAIT. It is up to the program to detect the value and act appropriately.

Examples

1 WAIT FOR 3

Waits for 3 seconds.

2 WAIT FOR "10:20"

Wait until 10:20 am.

3 WAIT x\$

Waits for a single key stroke and puts it in x\$.

4 WAIT MENU OR KEY

GET a\$

IF a\$ = "" THEN GOSUB menucheck ELSE GOSUB keycheck

Waits for a key stroke or for an item to be selected from a user-defined menu. You could use a routine like this to provide keyboard alternatives for your menu items. If no key has been pressed, the program assumes a menu item has been selected and branches to the subroutine which checks menu selections; otherwise it branches to the subroutine which checks for keyboard selections.

See Also

MENU, MOUSE ON/OFF, PANEL ON/OFF

WAIT COMMS

Purpose

Suspends Superbase operations until DCD line is active.

Syntax

WAIT COMMS

Comments

Superbase will poll the RS232 port once a second until the DCD line is determined to be active, i.e. a connection has been established. While not actually polling, the Superbase suspends operations.

See Also

OPEN COMMS

WHERE

Purpose

Set the filter conditions for a query or a report.

Syntax

WHERE *conditions*

Comments

WHERE is the program equivalent of the Filter command line in a query definition, and can only be used within a section that is headed by the query SELECT command. You can use WHERE to set a filter on the fields selected for report output or for other query applications, such as sorting, merging files, or simply retrieving data with query that has been saved to disk.

conditions takes the same form as the Filter command line in the query definition requester. WHERE is a multi-file filter command – unlike the record selection command SELECT WHERE – and if it is used for this purpose, the join between two files must be placed at the beginning of the statement, as in:

WHERE Lastname.Clients = Lastname.Deposits

Any subsidiary conditions can then be added to the line using the AND operator.

The WHERE keyword may be used as a system variable. It returns the filter for the current file as a string. This allows a program to analyze the results of user input during a SELECT WHERE ASK command.

```
SELECT WHERE ASK
x$ = WHERE
IF x$ LIKE "**field1*" THEN fld$(1) = "field1"
```

Examples

- 1 SELECT Firstname.Clients, Lastname.Clients, Bank, Amount
WHERE Lastname.Clients = Lastname.Deposits AND
Lastname.Clients LIKE "[d-e]*"
ORDER Lastname.Clients
END SELECT
- 2 SELECT Stockcode, Description, Price
WHERE Price >= 50 AND Price <= 100

Notes

In the first example, WHERE is used to set up a multi-file filter. It selects those clients whose details are also stored in the Deposits file and whose last name initial is D or E.

See Also

SELECT

WHILE WEND

Purpose

Executes a series of instructions as long as the specified conditions are true.

Syntax

```
WHILE exp statements WEND
```

Comments

WHILE and WEND set up a loop, in which the statements in between are executed repeatedly for as long as the expression following WHILE is true. When the expression is not true, execution resumes with the first statement after WEND.

WHILE WEND statements may be nested.

Examples

- 1 OPEN FILE "Address"
SELECT FIRST
WHILE NOT EOF("Address")
VIEW
SELECT NEXT
WEND
- 2 WHILE NOT EOF("**"): INPUT &1, a\$: ? a\$: WEND
- 3 WHILE NOT EOF("Myfile")
IF Field1.MYFILE > 1000 THEN END WHILE
SELECT NEXT
VIEW
WEND

Notes

Example 1 opens a file and then displays each of the records in turn (equivalent to clicking on the fast forward button). Example 2 reads a text (sequential) file line by line, displaying each line on screen.

Example 3 shows how the command END WHILE may be used to achieve a controlled exit from a WHILE WEND block. Control is transferred to the program statement following WEND.

See Also

FOR ... TO ... NEXT

YEAR

Purpose

Returns a numeric value for the year from a julian date number.

Syntax

YEAR (*expr*)

Comments

The function is only valid for dates from January 1, 1 A.D. to December 31, 9999. Consequently *expr* is only valid in the range 1 to 3652048. If *expr* is 0 then the number returned is 0. If *expr* is negative the results are unpredictable.

Associated date functions are DATE\$ DAY DAYS DAY\$ MONTH MONTH\$

Examples

- 1 numfieldc = YEAR(datefielda)
- 2 numfieldc = YEAR(datefielda + 90)
- 3 numfieldc = YEAR(TODAY)
- 4 x%% = YEAR(datefielda + VAL(textfielda))
- 5 x%% = YEAR(DAYS("Jan 11 85"))
- 6 ? YEAR(datefielda + 30)

Notes

Example 3 provides a calculation to insert the year number of the system date into a numeric field.

See Also

DAY, MONTH

18 DML SUMMARY

Application Functions

CALL *strexpr* [,*nexpr*]

Execute AmigaDOS command, open CLI window or communicate with ARexx

CHAIN *filename*

Execute program without clearing variables

EDIT [TEXT/QUERY/UPDATE]

Allow user to edit program, text, query or update

LOAD [TEXT/KEY/QUERY/UPDATE] *filename* [,APPEND]

Load program, text, file,function key list, query or update

NEW [TEXT/QUERY/UPDATE]

Clear program or text area

PROTECT [*filename*]

Save the current program in encrypted form

RUN [*filename*]

Execute program, optionally loading from disk

SAVE [TEXT/KEY/QUERY/UPDATE] *filename* [,TEXT]

Save program, text file, function key list, query or update

System Commands

BREAK [ENTER] ON/OFF

Set or clear user stop enabled

COPY *from filename* [,/TO] *to filename*

Copy any system file

DATEBASE *string*

Set default date format

DEBUG ON [,*speed* / OFF

Allows single step execution of a DML program

DELETE *filename*

Delete any system file

DIRECTORY *path*

Change directory to path.

KEY *keynum* [,*string*]

Set keynum to string or clear it

LIST *filename*

List any system file to screen

MOUSE ON / OFF

Switch mouse pointer shape from hourglass to arrow

NUMBASE *string*

Set default numeric format

PANEL ON / OFF

Switches the browsing controls on or off

QUIT

Exit Superbase system

RENAME *old filename* [,/TO] *new filename*

Rename any system file

SET BUFFERS *nexpr*

Set number of cache buffers to use

SET EDIT ON/OFF

Set auto-entry to program editor

SET EDIT ATTR [ON/OFF]

Determine color attributes of field data during data entry

SET FIELDS ON/OFF

Determine whether fieldnames are shown in default views

SET FN KEY ON/OFF

Allows detection of function key press within DML program

SET HEADING *stexpr*

Set Superbase window title bar

SET NOW

Set system variable NOW

SET PAGE

Set default view

SET PAGING [ON/OFF]

Set paging

SET PG *stexp*

Set the margins on the printer

SET PG REQUEST

Invokes Page Setup requester

SET POSITION

Set position and size of windows

SET PRINTER

Load printer control file for specified printer

SET PRINTER REQUEST

Invokes printer driver requester

SET PRINTER *strexpr*

Select printer

SET REQUEST [ON/OFF] / *nexpr*

Specifies confirmation messages

SET REQUEST HEADING

Set custom title for Superbase system requesters

SET STATUS *strexpr*

Places a message in the status line

SET TABLE/RECORD/PAGE/FORM

Set view according to parameter

SET TEXT *strexpr*

Select font

SET TODAY

Set system variable TODAY

STATUS [ON/OFF]

Turns status line display on or off

Basic Statements**CLEAR**

Clear all system variables

DATA *constant* [,*constant*]...

Specify data for READ statement

DIM *array variable*

Set array dimensions

ERASE *varlist*

Remove a variable from memory

EXECUTE *string*

Execute text string as though command

[LET] *var/field* = *exp*

Assign value of expression to variable or field

READ *var/field* [,*var/field*]

Read data into variables or fields from data pointer

REM *text*

Non executable comment line

RESTORE [*label*]

Move data pointer to specified position or home

Flow Control

END

Terminate execution of application

FOR *var* = *nexpr* TO *nexpr* [STEP *nexpr*] *statements* NEXT [*var*]

Repeat program lines a number of times

GOSUB *label*

Call a procedure or subroutine

GOTO *label*

Branch to the specified label

ON ERROR [GOTO *label*]

Specify procedure to be followed on error condition

ON *nexpr* GOSUB *label* [,*label*]...

Call procedure or subroutine in list

ON *nexpr* GOTO *label* [,*label*]...

Branch to statement or label in list

RESUME [NEXT / *label*]

Resume execution after error at next or specified position

RETURN

Return from procedure or subroutine execution

WEND

Mark end of while command sequence

WHILE *expr*

Perform following commands if expression true

Conditionals

IF *expr* THEN *statements* [ELSE IF *expr* THEN *statements*] [END IF]

[ELSE *statements*] [END IF]

Conditional statement execution

SELECT CASE *test expr*

CASE *expressions statements*

[CASE *expressions statements*]

[CASE ELSE *statements*]

END SELECT

Conditional statement execution

varfield = expr ? expr : expr

Conditional assignment of value to field or variable

File and Index Commands

ADD [FILE *sbfname*] *field definition string* [*formula string*]

Add a new field to a file

CLOSE [ALL]/[FILE *sbfname*]

Close all or specified files

CLOSE FIELDS [FILE *sbfname*]

Close the open fields for specified file

CREATE FILE *sbfname* AND *sbfname* FROM *sbfname*

Split database file into two files

CREATE INDEX ON *exp* [FILE *sbfname*] [UNIQUE]

Create a new index file optionally make unique

CREATE NEW FILE *sbfname* FROM FILE *sbfname*

Derive empty file from existing file

CREATE *sbfname*;passwords

Create a new database file in memory

FILE *sbfname*

Select the default file to be used

INDEX *index* / *strexpr*

Select the default index to be used for a file

MAKE *sbfname*

Make the file exist on disk and store the file def

MODIFY *field* [,] *field definition string* [*formula string*]

Modify parameters for field changing name, type etc.

OPEN FILE *sbfname*;passwords

Open file set as default

OPEN INDEX *dbasefile*.NDX

Open dBase index for use with dBase file

OPEN FIELDS [FILE *sbfname*] [*fieldlist*] / [ADD *fieldlist*]

Open a set of fields for specified file

PASSWORD *sbfname*; passwords

Set new passwords for a specified file

REMOVE FILE *sbfname*

Remove all data, file and all indexes

REMOVE INDEX *index*

Remove the specified index for specified file

REORGANIZE [FILE *sbfname*] [TO] *pathname*

Reorganize current or specified file to new pathname

SAVE FILE *sbfname*

Save the current file definition

SET FILE MAX ON/OFF

Set extended file handling on or off

Record Commands

BLANK [FORM] / [DUPLICATE] [FILE *sbfname*]

Create new record, optional duplicate

ENTER [*field1* [ROW *nxpr1*]] / [*nxpr2*]

[TO *field2* [ROW *nxpr3*]] / [,*nxpr4*]

Allow the user to enter a record in the current file or form

SELECT

Record selection commands

SELECT *field parms* [WHERE *parms*] [REPORT *parms*] [ORDER *parms*]

[TO PRINTER/*filename*/FILE *sbfname*] [END SELECT]

Query Language Command.

SELECT [FORM] CURRENT [FILE *sbfname*] [INDEX *index*]

Select the current record in key sequence

SELECT [FORM] DUPLICATE [FILE *sbfname*] [INDEX *index*]

Select the next record with the same key

SELECT [FORM] FIRST [FILE *sbfname*] [INDEX *index*]

Select the first record in key sequence

SELECT [FORM] KEY *string* [FILE *sbfname*] [INDEX *index*]

Select the first record with index matching *string*

SELECT [FORM] LAST [FILE *sbfname*] [INDEX *index*]

Select the last record in key sequence

SELECT [FORM] NEXT [FILE *sbfname*] [INDEX *index*]

Select the next record in key sequence

SELECT [FORM] PREVIOUS [FILE *sbfname*] [INDEX *index*]

Select the previous record in key sequence

SELECT [FORM] REMOVE [FILE *sbfname*]

Remove the current record in selected file

SELECT [FORM] [WHERE [FILE *sbfname*] [*conditions*]] / [ASK]

Select the first record matching conditions or clear conditions

STORE [FORM]/[FILE *sbfname*]

Store current record in the file

VIEW

View record in the current file

Process Commands

EXPORT [FILE *sbfname*] [INDEX *index*] [TO] *filename*

[WHERE *conditions*]/[ASK] [USING *parms*]

Export to external file

IMPORT *filename* [[TO] FILE *sbfname*] [WHERE *conditions*]/[ASK]
[USING *parms*]

Import external text file to Superbase

LABELS [FILE *sbfname*] [WHERE *conditions*] [USING *labelstring*]

Print labels as per label definition

MERGE [TEXT *filename* [WHERE *conditions*]

Load text file and mail merge

PRINT CURRENT [PAGE/ALL] / FILE [PAGE/ALL] *sbfname* [INDEX
indexname] [DESCENDING] [WHERE *conditions* / ASK] / [*expressionlist*]

Sends information to the printer

REMOVE FROM FILE *sbfname* WHERE *conditions*

Remove records matching conditions

UPDATE *calc list* [WHERE *conditions*] [END UPDATE]

Perform relational update

Input Output Functions

? DIRECTORY

Current directory listing

? /DISPLAY/PRINT/OUTPUT TO *file*

Send information to selected output device

? *exprlist*

Any expression list

? LIST

Program listing in memory

? MEMORY

List of variables in memory

? QUERY [TO PRINTER/FILE] *filename/filename*

Runs the current query to optional device

? STATUS [FILE *sbfname*]

Status of selected file or system

? TEXT [MERGE]

Text file in memory optionally mail merging

ASK [*string*] [*pos*] [*length*] ;*varfield*

Get input string from user

ATTR OFF

Clear italics and underline

BELL

Flash screen

BF [ON / OFF]

Set or clear boldface

CLOSE INPUT/OUTPUT

End input/output to/from text file

CLS

Clear output screen

EJECT [*nexpr*]

Do new pages on print device

GET *varfield*

Get character from keyboard no wait

HOME

Move screen output position to home

INPUT [&*nexpr*/LINE] *varfield*

Input characters or line from text file

IT [ON / OFF]

Set or clear italics

LOCATE *coordinates*

Set position on output device

MEMORY

List of variables in memory

MENU CLEAR

Clear user-defined menus

MENU *column, item, state* [,*text*]

Set up a user defined menu

MENU ON *numvar*

Turn on user-defined menus, specify numeric variable for return value

MOUSE *nvar1, nvar2, nvar3*

Reads the position of the mouse pointer and responds to mouse button input

NEWLINE [*nexpr*]

Send newline to output device

OPEN *filename* FOR INPUT/OUTPUT/APPEND

Open sequential file for input/output

POSITION *nexpr*

Position in sequential file

REQUEST *text1*[, *text2*[, *type* [, *numvar* [, *textvar* [, *len*]]]]

Set up a user-defined requester

SET *filename*

Read exec or variable file and execute

SHOW *field* / *strexpr*

Show external field

STATUS [FILE *sbfname*]

Status of selected file or system

WAIT [MOUSE/MENU/KEY/PANEL [OFF]] / [FOR *time/nexpr*] / [*var/field*]

Wait for specified time or until key pressed or mouse button clicked

UL [ON / OFF]

Set or clear underline

Reporting

AFTER GROUP *statements*

Specify after group activity

AFTER REPORT

Specify after report activity

AFTER SELECT

Specify activity on reading record

BEFORE GROUP *statements*

Specify before group activity

BEFORE REPORT

Specify before report activity

BEFORE SELECT

Specify activity before reading record

END GROUP

End of group specifications

END REPORT

End of report specifications

FOOTING *statements* END FOOTING

Specify statements to execute on page footing

GROUP *field, totallist*

Specify subtotal break field and subtotals, means and counts for group

HEADING *statements* END HEADING

Specify statements to execute on page heading

REPORT *total list*

Set report totals, means and count

Form Handling**ADD FORM**

Add object to form

ADD FORM REPLICATE

Add transaction block object to form

CLOSE FORM

Close current form

CREATE FORM

Create new form in memory

CREATE PAGE

Create new form page in memory

ENTER [*field* [ROW *nexpr*]] / [*nexpr*] [, *nexpr*]

Enter data into fields through current view or form

FORM [, *page* [, *row* [, *column*]]]

Specify page for current and top left-hand corner

OPEN FORM *formname*

Load a form into memory

SELECT FORM NEXT/PREVIOUS PAGE

Read and display the next or previous set of transaction records on a one-to-many relational form.

SELECT FORM ROW *nexpr*

Select current line in transaction form

SET FORM ATTR/BF/BG/FG/IT/PAGE/TEXT/UL/USING

Set attributes of form objects or text or fields

UPDATE FORM ROW *nexpr*

Copies data from current record to transaction line

Comms Commands

CLOSE COMMS

Close the communications channel

COMMS ? *strexpr*

Output characters through the comms channel.

COMMS FILE [?] *strexpr*

Transmit a file using the comms channel.

COMMS FILE GET *strexpr*

Receive a file using the comms channel.

COMMS GET [TEXT] *stringvar*

Read a character from the comms buffer.

COMMS INPUT [TEXT] *stringvar*

Read characters from the comms buffer

OPEN COMMS [USING *parameters*]

Open the communications channel

WAIT COMMS

Suspend Superbase operations until DCD line is active

Operators

Arithmetic Operators

$\wedge$	Exponentiation
$-$	Negation
$*$	Multiplication
$/$	Division
MOD	Modulo arithmetic
$+$	Addition
$-$	Subtraction

Relational Operators

$=$	Equality
LIKE	Pattern matching case insensitive equality
CONTAINS	Pattern matching for text files
$\langle \rangle$	Inequality
$<$	Less than
$>$	Greater than

<= Less than or equal to
>= Greater than or equal to

Logical Operators

NOT AND OR

Mathematical Functions

ABS(*nexpr*)

Absolute value of variable

ATN(*nexpr*)

Arctangent

COL(0)

Return current screen column

COS(*nexpr*)

Cosine

DISKSPACE("*disk*")

Return free disk space

EOF(*sbfname*)

Return if at end of file

EXISTS (*sbfname* [*indexfield*])

Check whether a file exists on disk or whether an index value exists in an index

EXP(*nexpr*)

Exponent

FIX(*nexpr1*,*nexp2*)

Fix decimal precision of value

FN fact(*nexpr*)

Returns the factorial of a number

FOUND(*sbfname*)

Return result of last search

FREE(*nexpr*)

Return free memory size

INT(*nexpr*)

Integer portion of variable

LOG(*nexpr*)

Logarithm

LOOKUP(*value/field1 field2*)

Return if value exists in file (indexed field)

PCOL(*nexpr*)

Return current printer column

PROW(*nexpr*)

Return current printer row

RECCOUNT(*sbfname*)

Return number of records in file

RND(*nexpr*)

Random number

ROW(0)

Return current screen row

SER(*sbfname*)

Return serial number of specified file

SGN(*nexpr*)

Sign of variable

SIN(*nexpr*)

Sine

SQR(*nexpr*)

Square root

TAN(*nexpr*)

Tangent

String Functions

ASC(*strexpr*)

Ascii value of character

CHR\$(*nexpr*)

String value of character

DATE\$(*nexpr*[, *format string*])

Date string from numeric using optional format

DAY(*date*)

Numeric day value of date

DAY\$(*date*)

Day of week from date

DAYS(*strexpr*)

Numeric value of date

ERR\$(*nexpr*)

Returns error message for error number *nexpr*

FCASE\$(*strexpr*)
Capitalize first letter of string

FORMAT\$(*strexpr*, *nexpr1* [, *nexpr2*])
Forces query and report output to wordwrap within a column.

HRS(*time*)
Number of hours from time

INSTR([*nexpr*], *strexpr1*, *strexpr2*)
Find position of substring *strexpr2* in *strexpr1*

LCASE\$(*strexpr*)
Convert string to lower case

LEFT\$(*strexpr*, *nexpr*)
Left portion of string

LEN(*strexpr*)
Length of string

LTRIM\$(*strexpr*)
Trim leading spaces from *strexpr*

MID\$(*strexpr*, *nexpr* [, *nexpr*])
Mid portion of string

MINS(*time*)
Number of minutes from time

MONTH(*date*)
Numeric month value of date

MONTH\$(*date*)
Month string from date

PAD\$(*strexpr* [, *nexpr*])
Pads a string or text field with spaces

REPLICATE(*strexpr*, *nexpr*)
Replicate character expression *nexpr* times

RIGHT\$(*strexpr*, *nexpr*)
Right portion of string

SECS(*time*)
Number of seconds from time

SPACE\$(*nexpr*)
Return string with *nexpr* spaces

STR\$(*nexpr* [, *nexpr1*] [, *nexprd*] / [, *format string*])
String from number with optional format

THOUSECS(*time*)

Number of thousandths of second from time

TIME\$(*nexpr*[*format string*])

Time string from numeric using optional time format

TIMEVAL(*time*)

Numeric value of time

TRIM\$(*strexpr*)

Trim trailing spaces from *strexpr*

UCASE\$(*strexpr*)

Convert string to upper case

VAL(*strexpr*)

Value of string

YEAR(*date*)

Numeric year value of date

FN alpha(*strexpr*)

Returns a string containing only alphabetic characters (a-z, A-Z)

FN ansi(*strexpr*)

Returns a string in which all IBM characters are converted to the ANSI character set

FN dec (*strexpr*)

Returns the decimal equivalent of a string into a number

FN ext (*strexpr*)

Returns a string containing the extension of a filename

FN hex (*nexpr*)

Returns a string containing the hexadecimal equivalent of a decimal

FN ibm(*strexpr*)

Returns a string in which all ANSI characters are converted to the IBM character set

FN name (*strexpr*)

Returns a string containing a filename together with its extension

FN numeric(*strexpr*)

Returns a string containing only numeric characters (0-9)

FN path (*strexpr*)

Returns a string containing all pathname characters including final backslash

FN root (*strexpr*)

Returns a string containing the root of a filename

Financial Functions

FN fv(*rate,nper,pmt[,pv[,type]]*)

Returns the future value of an investment

FN nper(*rate,pmt,pv[,fv[,type]]*)

Returns the number of payments on an investment

FN pmt(*rate,nper,pv[,fv[,type]]*)

Returns the value of a periodic payment

FN pv(*rate,nper,pmt[,fv[,type]]*)

Returns the present value of an investment

FN rate(*nper,pmt,pv[,fv[,type[,guess]]]*)

Returns the interest rate per period for an investment

FN sln(*cost, salvage, life*)

Returns straight line depreciation for an asset

FN sys(*nexpr*)

Returns system information

Variables

ARRAYS

X%/%/X&%/X#%/X!%/X%/X\$ (*nexpr[[,nexpr][, nexpr]]*)

DIRECTORY

Return path of the current directory.

ERRNO

Return current error number

FIELDS BY NAME

Fieldname ; Field.file ; Field."file"

FILE

Return name of current file

FORM

Return name of the current form (or a null string if no form is open)

INDEX

Return name of current index

MULTIPLE RESPONSE FIELDS

Fieldname(*nexpr*)

NOW

Return system time

NUMERIC VARIABLES

X%/%/X&%/X#%/X!%/X%

ORDER

Return the data entry order number of the most recently edited field on a form

PG

Return current page number in report

PI

Return value of pi

POSITION

Returns the position of the data pointer in an ASCII file

STRING VARIABLES**X\$****TODAY**

Return system date

—

—

.

—

19 USING AREXX WITH SUPERBASE

One of the benefits of the Amiga's multi-tasking operating system is its ability to run different applications at the same time. This allows you to switch easily between different applications such as database and spreadsheet, without having to quit from one in order to run the other. However, even greater productivity may be achieved if the different applications can be actively integrated rather than just co-existing. Such integration is possible through Superbase's support for the ARexx programming language, which is available both as part of the Workbench 2.0 System Software and separately.

This chapter assumes that you are familiar with ARexx as documented in that system's user guides.

The ARexx macro language is well suited to the multi-tasking environment. It allows you to create integrated systems in which separate applications pass data and commands between each other: a feature known as Inter-Process Communication, or IPC. Not every Amiga application supports ARexx, but both Superbase and Precision's spreadsheet system, Superplan, provide all the necessary features. Through ARexx, an ARexx program or another application that supports ARexx can instruct Superbase to execute any valid Superbase DML command. Likewise, Superbase can pass command messages to other applications or run programs written in the ARexx language itself.

ARexx Ports

ARexx communicates via ports. Each application may have one or more ports, to which messages may be sent. ARexx's own default port is called REXX. (Note that ARexx is case sensitive.)

Superbase Professional 4's port is called SBpro4. The ports of second and subsequent instances of Superbase are identified with a suffix of an underscore followed by a number, such as SBpro4_2.

The Superplan ARexx port is called SpRexx. Superplan does not support multiple ports.

Communicating with Applications from Superbase

To communicate either with ARexx itself or with another application, you use the DML command CALL. You use different syntax depending on whether you are sending commands to an application or requesting data from it.

Sending Commands

To send a command to another application, use the following DML syntax:

```
CALL portname EXECUTE strexpr
```

The *portname* must be a string expression naming an ARexx port. The string *strexpr* must consist of commands intelligible to the other application (maximum length 255 characters). Normally this is equivalent to a string of characters typed at the keyboard.

For example:

```
CALL "SpRexx" EXECUTE "/rrC1,D1:G1ENT"
```

This command addresses the ARexx port of Superplan and causes the execution of a Superplan command that replicates cells across the top row of the worksheet.

Requesting Data

To obtain data from another application, use the following DML syntax:

```
CALL portname RETURN strexpr [TO field/var]
```

As above, *portname* must name a valid ARexx port.

The *strexpr* must refer to a data item intelligible to the other application, such a spreadsheet cell reference. For example,

```
CALL "SpRexx" RETURN "c23" TO cell$
```

would request that the contents of cell c23 in the active Superplan worksheet be returned in the Superbase variable cell\$.

Running ARexx programs

You may need to launch actual ARexx programs from within Superbase. To do this, use the CALL...RETURN syntax as explained above. ARexx programs are designed to return exit codes to the applications that call them. The exit code of an ARexx program called from Superbase is placed in the *strexpr* that normally receives returned data. For example,

```
CALL "REXX" RETURN "my_ARexx_prog" TO exitcode$
```

would run the ARexx program my_ARexx_prog and return its exit code in the Superbase variable exitcode\$.

Superbase does not execute ARexx programs asynchronously; it waits until the ARexx program terminates before continuing DML program execution.

ARexx programs can send results back to Superbase by setting variables through Superbase's own port, as described on the next page.

Controlling Superbase from ARexx

As explained above, any valid DML command may be passed to Superbase for execution. This example shows how an ARexx macro makes Superbase select the first record in a file and then obtains the value of the Lastname field in that file:

```
/* An ARexx macro */  
address SBpro4 "SELECT FIRST"  
options results  
address SBpro4 "Lastname.CLIENTS"  
say RESULT
```

The result of an action will be returned in the ARexx system variable RESULT.

Error Handling

If ARexx is unable to complete a requested operation, it returns an error value to Superbase in the DML global variable ERRNO, which is described in full in Volume 2, Chapter 17. The Superbase DML program that calls ARexx must check ERRNO before and after the CALL command to see whether it succeeded. No internal Superbase error is generated in this situation.

When an ARexx program calls Superbase, Superbase always returns an error code. If no actual error occurs during the execution of the DML command Superbase returns 0. If the DML command does generate an error in Superbase, Superbase returns its own error number. This can be examined in the ARexx system variable RC.

Busy Status

If ARexx or another application calls a Superbase Professional port while Superbase is busy processing, either in a DML program or a menu operation such as Process Query or Record New, Superbase returns the value -1 to ARexx or the calling application. The application must retry until Superbase is freed up, or abandon the communication.

Index

!

& 17-4 - 17-5

? 17-7

? Commands 17-3

? DIRECTORY 17-8

? LIST 17-9

? MEMORY 17-9

? QUERY 17-10

? STATUS 17-11

? STATUS FILE 17-11

? TEXT 17-12

@ 17-4 - 17-5

A

ABS 17-13

ADD 17-14

ADD FORM 17-17

ADD FORM REPLICATE 17-19

AFTER GROUP 17-20

AFTER REPORT 17-21

AFTER SELECT 17-21

ALL 17-6

APPEND 17-112, 17-135

Arithmetic operators 15-8

Arrays 15-3

ASC 17-22

ASCENDING 17-142

ASK 17-23

As a parameter 17-187

ATN 17-24

ATTR 17-4

B

Batch mode 17-211

Before Group 10-8, 17-25

BEFORE SELECT 17-25

BELL 17-26

BG 17-4

BLANK 17-27

BREAK 17-28

BREAK ON/OFF 17-28

C

CALL 17-29

CHAIN 17-30

Changing the output device 17-3

Check boxes 9-4

Defining 9-5

Editing 9-5

CHR\$ 17-30

CLEAR 17-31

CLOSE 17-34

CLOSE COMMS 17-32

CLOSE FIELDS 17-32

CLOSE FILE 17-33

CLOSE FORM 17-33

CLS 17-35

COL 17-35

Command line 16-5

Editing 16-6

COMMS ? 17-36

COMMS FILE 17-36

COMMS FILE GET 17-37

COMMS GET 17-38

COMMS INPUT 17-38

Constants 15-13

CONTAINS 15-11

COPY 17-39

COS 17-40

COUNT 17-40

CREATE 17-41

CREATE FORM 17-42

CREATE INDEX 17-44

CREATE PAGE 17-44

D

DATA 17-45, 17-155

- Date 15-5
- Date format 15-5
- DATE\$ 17-46
- DATEBASE 17-47
- DAY 17-48
- DAY\$ 17-49
- DAYS 17-50
- dBase conversion 17-68
- dBase files
 - Indexes 17-141
 - Opening 17-139
- DEBUG 17-51
- Defining the object hierarchy 3-5
- DELETE 17-51
- DESCENDING 17-142
- Directories 12-1
- DIRECTORY 17-53
 - See ? DIRECTORY
- Directory List 12-1
- DISKSPACE 17-53
- DISPLAY 17-54
- DOWN 17-4, 17-6

E

- EDIT 17-55
- Editing Text 16-5
- EJECT 17-7, 17-56
- END 17-56
- END FOOTING 17-57
- END GROUP 17-57
- END HEADING 17-58
- END REPORT 17-58
- ENTER 17-59
- EOF 17-62
- ERASE 17-63
- ERR\$ 17-64
- ERRNO 17-64
- Error trapping 17-133
- EXECUTE 17-65
- EXISTS 17-66
- EXP 17-67
- EXPORT 17-68
- Expressions 15-13
- External files 17-208

F

- FCASE\$ 17-71
- FG 17-4
- FIELD 17-176
- Field names 15-4
- Fields 15-4
 - Definition 17-14
 - Types 17-14
- FILE 17-71
- File definition 17-14
- File types 15-4
 - Program 16-7
- Filters 10-14
- FIX 17-72
- FN alpha 17-74
- FN ansi 17-74
- FN dec 17-75
- FN ext 17-75
- FN fact 17-76
- FN fv 17-76
- FN hex 17-77
- FN ibm 17-78
- FN name 17-78
- FN nper 17-79
- FN numeric 17-79
- FN path 17-80
- FN pmt 17-80
- FN pv 17-81
- FN rate 17-82
- FN root 17-82
- FN sln 17-83
- FN sys 17-84
- FOOTING 17-85
- FOR TO NEXT 17-85
- FORM 17-87
- Form commands 9-1
 - Auto execution 9-3
 - Without pushbuttons 9-3
- Form creation commands
 - ADD FORM 17-17
 - ADD FORM REPLICATE 17-19
 - CREATE FORM 17-42
 - CREATE PAGE 17-44
 - SET FORM ATTR 17-194
 - SET FORM BF 17-195

SET FORM BG 17-196
SET FORM FG 17-196
SET FORM IT 17-197
SET FORM PAGE 17-198
SET FORM TEXT 17-198
SET FORM UL 17-199
SET FORM USING 17-199
Summary 15-19
Form creation under DML 15-19
FORMAT\$ 17-88
Formatting numeric output 17-5, 17-72
Formatting report output 10-12
Forms
 Programming 9-4
 Removing 12-2
FOUND 17-89
FREE 17-90
Function keys
 Load 17-112
 Save 17-173
Functions 15-6

G
GET 17-90
GOSUB 17-91
GOTO 17-92
GROUP 17-93

H
HEADING 17-94
HOME 17-94
Housekeeping 12-1
HRS 17-95

I
IF THEN ELSE 17-96
Image file types 2-3
IMPORT 17-99
INDEX 17-102
INPUT 17-103
INSTR 17-104
INT 17-105
IT 17-4, 17-6

K
KEY 17-105
Key Controls
 Program Editor 16-4

L
Labels 15-14, 17-107
LCASE\$ 17-108
LEFT\$ 17-109
LEN 17-110
LET 17-110
LIKE 15-11
Line format 15-14, 16-3
LIST 17-112
LOAD 17-112
LOCATE 17-113
LOG 17-114
Logical operators 15-11
LOOKUP 17-115
LTRIM\$ 17-116

M
MAKE 17-117
MAX 17-118
MEAN 17-118
MEMORY
 See ? MEMORY
MENU 17-119
MERGE 17-12, 17-122
MID\$ 17-123
MIN 17-124
MINS 17-124
MOD 17-125
MODIFY 17-126
MONTH 17-126
MONTH\$ 17-127
MOUSE 17-128
MOUSE ON/OFF 17-129

N
Nesting
 FOR TO NEXT 17-87
 IF THEN ELSE 17-99
 SELECT CASE 17-179
 WHILE WEND 17-226

NEW 17-130
NEWLINE 17-4, 17-6, 17-130
NOW 17-131
NUMBASE 17-132
Numeric format 17-5, 17-73, 17-132
Numeric variables 15-3

O

Object hierarchy 3-5
Object types
 Check boxes 9-4
 Commands 9-1
 Radio buttons 9-6
ON ERROR 17-133
ON GOSUB 17-134
ON GOTO 17-135
OPEN 17-135
 FOR INPUT 17-136
 FOR OUTPUT 17-135
OPEN COMMS 17-137
OPEN FIELDS 17-138
OPEN FILE 17-139
OPEN FORM 17-140
Operating modes 15-1
Operators 15-8
 Precedence 15-12
OR 17-223
Order 17-141
 Transactions 4-4
Output device
 Disk 17-3
 Printer 17-3, 17-151
 Screen 17-3
Output format parameters 17-4, 17-176
OUTPUT TO 17-145
Output to disk 17-177
Outputting text files 17-12

P

Page Hierarchy 3-5
Page menu
 Remove 12-2
Page number 17-94
Page Remove 12-2
PANEL 17-147

PASSWORD 17-147
PCOL 17-148
PG 17-94
POSITION 17-149
PRINT 17-151
Printing data 17-151
Printing forms 17-151
Program
 Moving the insertion point 16-4
Program menu 16-1
Programs
 Creating 16-2
 Editing 16-4
 Loading 16-7
 Running 16-8
 Saving 16-7
Project
 Directory List 12-1
 Remove Form 12-2
 Status 12-1
Project menu
 Remove 12-2
PROTECT 17-153
PROW 17-154
Pushbuttons 3-5, 9-1
 Editing 9-2
 Using 9-2

Q

Query
 Load 17-112
 REPORT 15-17
 Save 17-173
 Section 15-17
 SELECT 17-176
 WHERE 15-17, 17-225
Query Language Commands 15-17
QUIT 17-154

R

Radio buttons 9-6
 Editing 9-7
READ 17-155
RECCOUNT 17-155
Record selection 15-16

- REM 17-156
- REMOVE FILE 17-157
- REMOVE FROM 17-158
- REMOVE INDEX 17-158
- REORGANIZE 17-159
- REPLICATE 17-160
- REPORT 17-161
 - Report commands
 - AFTER GROUP 17-20
 - BEFORE GROUP 17-25
 - Report functions 4-6, 15-7
 - Report programs 10-14
 - Reports
 - Printing 10-14
 - Running 10-12
 - Saving 10-12
- REQUEST 17-162
- Reserved words 15-1
- RESTORE 17-167
- RESUME 17-168
- RETURN 17-169
- RIGHT\$ 17-170
- RND 17-170
- Rounding 17-72, 17-213, 17-221
- ROW 17-171
- RUN 17-172
- S**
- SAVE 17-173
- SAVE FILE 17-173
- SAY 17-174
- SD 17-174
- SECS 17-175
- SELECT 17-176
 - ADD 17-176, 17-188
 - CURRENT 17-180
 - DUPLICATE 17-180
 - KEY 17-184
 - LAST 17-185
 - NEXT 17-185
 - With a semicolon 17-143, 17-176
- SELECT commands 15-16
- SELECT FIRST 17-181
- SELECT FORM ROW 17-183
- SELECT PREVIOUS 17-186
- SELECT REMOVE 17-186
- SELECT WHERE 17-187
- SER 17-189
- SET 17-190
 - BUFFERS 17-191
 - FORM 17-207
 - PAGE 17-207
 - PAGING 17-201
 - PG 17-201
 - POSITION 17-203
 - PRINTER 17-203
 - PRINTER REQUEST 17-204
 - RECORD 17-207
 - TABLE 17-207
 - TEXT 17-206
 - View 17-207
 - View mode 17-207
- SET EDIT 17-191
- SET EDIT ATTR 17-192
- SET FIELDS 17-192
- SET FN KEY 17-193
- SET FORM ATTR 17-194
- SET FORM BF 17-195
- SET FORM BG 17-196
- SET FORM FG 17-196
- SET FORM IT 17-197
- SET FORM PAGE 17-198
- SET FORM TEXT 17-198
- SET FORM UL 17-199
- SET FORM USING 17-199
- SET HEADING 17-200
- SET NOW 17-200
- SET PG 17-201
- SET PG REQUEST 17-202
- SET REQUEST 17-204
- SET REQUEST HEADING 17-205
- SET TODAY 17-206
- SGN 17-208
- SHOW 17-208
- SIN 17-209
- Sorting data 17-142
- SPACE\$ 17-210
- Splitting database files 17-42
- SQR 17-211
- Start up program 16-8

Status 12-1
 See ? STATUS
STORE 17-211
STR\$ 17-213
String variables 15-3
Syntax 15-15
System status 17-11
System variables 15-3

T

TAN 17-215
Ternary operator 17-111
Test 10-11
Text
 Load 17-112
 Save 17-173
THOUSECS 17-215
Time 15-5
TIMES\$ 17-216
TIMEVAL 17-216
TO FILE 17-177
TO PRT 17-177
TODAY 17-217
Transactions 4-1
 Calculations 4-4
 Example 4-3
 Linking Files 4-4
 Programming 4-6
 Setting the entry order 4-4
 Table Form 4-7
TRIM\$ 17-217

U

UCASE\$ 17-218
UL 17-4, 17-6
Update 17-219
 Load 17-112
 Save 17-173
UPDATE FORM ROW 17-220

V

VAL 17-221
Validations 8-1 - 8-2
VAR 17-222
Variables 15-2

Arrays 15-3
Names 15-2
Numeric 15-3
String 15-3
System 15-3
VIEW 17-222

W

WAIT 17-223
WAIT COMMS 17-225
WAIT PANEL 17-147
WEND 17-226
WHERE 17-225
WHILE 17-226

Y

YEAR 17-227

CONTENTS

20 INTRODUCTION 20-1

21 THE MAILING SYSTEM 21-1

Controlling Mailit	21-1
Mailit: The Data	21-2
Mailit Forms	21-4
The MAILIT.SBP Program	21-5
Mailit Activities	21-8
Maintaining the Mailing List	21-8
Merging and Purging	21-11
Label Printing	21-18
Housekeeping Activities	21-19
Maintaining the RULEBASE	21-25
The Mailit Help System	21-29
Exit and General Purpose Routines	21-30

22 THE TRADING SYSTEM 22-1

The Database Structure	22-4
The Menu Structure	22-5
Files	22-8
Activities	22-22
Menu Selection	22-22
File Maintenance	22-31
Clients	22-31
Stocks	22-33
Transactions	22-34
The stkf Routine	22-35
Countries	22-41
Currencies	22-42
The Program STKFO	22-42
The Daily Activities Menu	22-47
Currency Rates	22-47
Stock Prices	22-47
Certificate Numbers	22-47
Cash Receipts and Payments	22-49
The Document Production Menu	22-53

Certificate Advices	22-53
Contract Notes	22-53
Statements	22-56
The Update Menu	22-56
New Highs	22-56
New Lows	22-57
Certificates Sent	22-57
Contract Notes Sent	22-58

23 PROGRAMMING IDEAS 23-1

Customizing the Superbase Environment	23-1
Adding Password Protection to Activities	23-3
Unattended File Transfer	23-5
Sending and Receiving Text	23-7
Importing Data from Non-ASCII Files	23-9
Formatting Output for Desktop Publishing	23-12
Formatting a Mail Merge File for WordPerfect™	23-16
Modifying a Program Produced by the Report Generator	23-19
Producing Statistics for Text Documents	23-25
Producing a Word List and Count for a Text File	23-28

INDEX I-1

20 INTRODUCTION

Superbase is a powerful and diverse system. It includes a wide range of tools for creating solutions to particular data management problems. You can input, edit and browse through data using views or forms, and exchange data between Superbase and other file formats. You can specify validations and calculations at the file definition or forms entry level. General-purpose output tools include filters, queries, forms and the report program generator, while specific facilities are provided for merging documents and producing mailing labels.

In addition to its presentation capabilities, the Superbase Form Designer allows logical elements like calculations, command calls and one-to-many relationships to be built into forms. In many situations, using this fourth generation power enables the greater part of a sophisticated application to be implemented without programming.

Utilizing Superbase as a development tool, you can combine various command strategies, from macros for often-repeated key or mouse sequences, through pull-down or list-type menus to function key sets and form pushbuttons.

Above all, Superbase provides a comprehensive database management language for accessing, fine-tuning and coordinating the use of all of its facilities.

Systematic coverage of the tools and facilities within Superbase is given in the first two volumes of the manual. This third volume sets out to give you a better understanding of the full potential of Superbase and to open up new possibilities by illustrating a variety of ways in which Superbase can be put to use to solve real problems.

The two chapters after the Introduction describe applications implemented with Superbase. The Mailing system described in Chapter 21 uses a simple file structure but some quite sophisticated processing in order to achieve its objectives. The use of DML within the Mailing application shows how programming can extend the power of the database. In fact, the most important aspect of this example is that it is entirely controlled by a single DML program.

A feature of the mailing system is the way that all its facilities are accessed from a main control menu screen. This screen shows a set of labeled pushbuttons, enabling you to click on the option you require.

The Trading system aims to fulfil a wider, more diverse set of requirements than the previous example, reflecting the typical mix of data entry, file maintenance document production and reporting needs of a commercial organization.

The Trading system employs a number of files, forms and programs, but like the Mailing system, all of the functions of the Trading system are accessed from a top-level menu, displayed by a menu control program. The simpler routines shown as options on the top-level menu are contained within this program, while the more

complex ones are structured in separate program modules. On completion, these separate programs always return control to the main program.

Also, the Trading system combines two distinct command strategies. At the top level there is a set of pull-down menus. The more complex activities accessible from these menus involve opening forms, and once a form is open there are pushbuttons on the form for activities like entering a new record, browsing, and so on.

By reading through these chapters you can gain an understanding of the flexibility of approach to problem-solving embodied within Superbase. This flexibility is fundamental to the design objectives of Superbase, reflecting as it does the diversity of problems and approaches that exist in the business and professional world.

The final chapter brings together a number of hints and programming examples that may provide the pointers on techniques that you are looking for, or help you see a problem in a new way.

The Runtime System

The Superbase Runtime System is the main vehicle for the distribution of turnkey Superbase applications. Any application entirely under program control, such as the Mailing or Trading systems described in this volume, can be set up to run with the Runtime system.

The Runtime system provides a way in which software developers can distribute applications written in DML, without each user having to own a full copy of Superbase itself. The Runtime system is easy to install, and includes some features specifically designed to assist software developers. As DML programs are automatically tokenized, the programmer need only transfer the DML modules to the target machine, where the Runtime system will be able to execute them as a self-sufficient application.

Copies of the Runtime System and further guidelines for its effective use are available under license directly from Precision Software as part of its Developer Programs. Please contact Precision Software for further information.

Program Listings

While every effort has been made to ensure that the program listings contained within this volume are accurate, they are provided for example purposes only, and no warranty of any kind is given either regarding their accuracy or their suitability for any particular purpose. Moreover the listings and descriptions contained herein may differ from the actual example programs and applications supplied on disk with this product. In the event of differences arising, the versions supplied on disk should be taken as superseding the versions described herein.

21 THE MAILING SYSTEM

Our first example, the 'Mailit' Mailing application, illustrates many features of the Superbase system. Although the file used as the heart of the database system is basically very simple, some of the processing is quite sophisticated, and you will find it helpful to understand how DML can extend the power of the database. In fact, the most important aspect of this example application is that it is designed to be entirely controlled by a DML program.

The purpose of Mailit is to simplify and speed up the maintenance of mailing lists. You may add records to the mailing list file, and edit or delete existing records. Utilities allow you to print labels, export ASCII data, backup the mailing list to another file, or restore a previously backed up file as the main mailing list.

A special file, RULEBASE, contains a set of conversion equivalents which you can use to standardize the contents of a mailing list – changing all occurrences of Drive to Dr., for example.

The most powerful aspects of Mailit are its merge and purge capabilities. Mailing list users frequently need to update their databases by purchasing fresh lists and merging them into an existing list. One of the hazards of this procedure is the inadvertent creation of duplicate records. It is not so bad if every detail in one record is the same as in another – this may be detected relatively easily. The problem arises when two records refer to the same person even though some of their details are different. How does the software figure out whether J. Doe is the same as John Doe, and how do John P. Doe and J. P. Doe fit in? Mailit has a clever routine that can be tuned to detect duplicate data in several ways, allowing you to purge out unwanted records either manually or automatically.

Controlling Mailit

The Mailit files and processing routines are all accessed from a main control menu screen. This screen shows a set of pushbuttons with the names of the system options, so all you have to do is click the relevant button to begin the activity you require.

This main menu screen and all the other aspects of Mailit are controlled by a Superbase DML program, MAILIT.SBP, which is examined later on in this chapter.

Mailit: The Data

The Mailit application is based on three simple data files: MAILBASE, RULEBASE, and HELPBASE. MAILBASE contains the names and addresses used for the actual mailing part of the application. RULEBASE contains a set of


```

Misc1          ;TXT          ;50          ;17 ;0   ;
Misc2          ;TXT          ;50          ;18 ;0   ;

```

The other interesting field in the mailbase file definition is Virfield, which is used in Mailit's merge and purge routines. Virfield uses a calculation formula to create a composite text string from elements of the zip, lastname, address1, company and city fields within the record. The resulting string is used to detect duplicate addresses within the database – if another record creates the same string, the probability that it is a duplicate of the first is significant. The sensitivity of the matching process may be tuned by the user, and one of Mailit's central activities is the comparison of records suspected of being duplicates, giving the user the option to delete unwanted records.

The RULEBASE File

When mailing lists are being merged, the data often comes from disparate sources. Data may be entered according to different conventions at different locations – one person will type "Avenue," another will type "Ave." The mailing application must provide a way of standardizing the data, both to improve the detection rate for duplicate records, and to rationalize what could soon become a pretty baroque set of possibilities.

```

FieldName      ;TXT          IXD ;15          ;0 ;0   ;
LookFor        ;TXT          ;15          ;1 ;0   ;
ChangeTo       ;TXT          ;40          ;2 ;0   ;
RuleNum        ;NUM          IXD ;99999.       ;3 ;0   ;

```

RULEBASE has one or more records for each field that may need searching for standardization. Each record stores a name of a field (Fieldname), the string to be searched for (LookFor), and its replacement string (ChangeTo). The RuleNum field is used for sorting the file.

The HELPBASE File

Finally, the HELPBASE file contains a set of text records which explain the application. Help screens are accessed by clicking the Help button provided on most of the entry and selection screens.

```

Topic          ;NUM          IXD ;99.          ;0 ;0   ;
HelpPage       ;DEL          ;99.          ;1 ;0   ;
Doc            ;TXT          ;L2000        ;2 ;0   ;

```

Each record in Helpbase has a single long text field and a topic number, which is used to look up the specific help topic at the appropriate points in the program.

Mailit Forms

Mailit uses a wide variety of forms, ranging from menu selection screens through data entry, parameter setting, and duplicate purging. The forms have been designed to be eye-catching, and it is worth remembering that color schemes may be easily altered with the Superbase Form Designer.

The Main Menu

The first form that the Mailit user encounters after running the application program, MAILIT.SBP, is the main menu screen form, MAILIT.SBV.

This screen presents a set of eight choices together with a help option. Each pushbutton initiates a processing routine. The Quit button returns you to Superbase's native menu environment.

Mailit always returns to this screen after completing the processing initiated from it.

Defining Merge and Purge Parameters

The Merge and Purge pushbuttons on the main menu use the same form for setting the parameters to be used in the subsequent processing.

The form illustrates the use of radio buttons to select options, in addition to pushbuttons for triggering the next stage of the routine. The user's choices are passed from the form to the main program, providing an easy point-and-click method of control, and eliminating the need for unnecessary typing.

The top left-hand set of radio buttons allow you to choose a preset 'level' of sensitivity for detecting duplicates in the MAILBASE file. A level of sensitivity is defined by the combination of values for the lengths of the fields used for matching records – the fields used are shown next to these radio buttons. If you want to alter one of the levels, you click the relevant preset radio button, then the Set Thresholds pushbutton, and type in new values for the fields. The program saves the presets automatically.

This screen also lets you choose whether to have all duplicates removed automatically or not. If you select the Manual option, Mailit uses the duplicate checking screen to show you each candidate for purging so you can decide whether to remove it or not.

Duplicate Checking

The duplicate checking screen presents two records at a time: a match master on the left, and a candidate for purging on the right.

Candidates are selected by Mailit using the detection parameters set in the previous form. The user has the choice of retaining the candidate as a separate record, replacing the master with the candidate, or deleting (purging) the candidate from the mailing list file.

The MAILIT.SBP Program

In this section we shall consider MAILIT.SBP purely in its role as a component of the overall system, leaving the detailed analysis until later in the chapter.

The Mailit application is designed to be controlled by a DML program. You cannot gain access to any of the processing routines such as merge or purge without running the program. You could enter data into the mailbase file using Superbase's Record New command, but you would not be able to make the most of the facilities built into the form by its designer. The same is true for a number of other activities, which may be available from the Superbase menus, but are more precisely controlled by the mailit program.

Controlling the application from a program also allows the designer to exclude any Superbase menu commands that are irrelevant to the application, or may even pose a threat to its security. For example, nowhere in Mailit can you gain access to file structures in order to modify them – in menu command terms, File Modify is not available. This ensures that the field names used in MAILIT.SBP cannot be changed; changing fieldnames would cause errors when the program was run. (Strictly speaking, you can get into Superbase with Mailit's Quit pushbutton, but it would be quite possible to remove this or else force Quit to leave Superbase altogether.)

Starting Mailit

To start the application, the user clicks on MAILIT.SBP on the Windows desktop, or runs the program from Superbase's DML menu.

MAILIT.SBP initializes itself, opens the files, and presents the main menu screen. Here is some of the initialization code:

Option Selection

```
ON ERROR GOTO Errtrap
DIM prs%(5,5)
PANEL OFF
SET STATUS "Setting up..."
SET "MDEFAULT"
pth$ = DIRECTORY
activity$ = ""
OPEN FORM "Mailit"
```

This routine reads in variables from the last session with SET "MDEFAULT", sets up error detection, and presets some of the variables used on the form.

This is the routine that sets up the system wide commands menu. It checks for keyboard presses, mouse clicks, and the selection of the menu bar. It also sets up the form.

```
MENU 1,1,1,"&Add to Mailing List"
MENU 1,2,1,"&Merge Lists"
MENU 1,3,1,"&Edit Mailing List"
MENU 1,4,1,"Print &Labels"
MENU 1,5,1,"&Utilities  "
MENU 1,6,1,"&Purge Duplicates"
MENU 1,7,1,"Set &Options"
MENU 1,8,1,"&Help      "
MENU 1,9,1,"&Quit       "
```

MainMenu:

```
badd$ = "Add to Mailing List"
bEdit$ = "Edit Mailing List"
bLabel$ = "Print Labels"
bMerge$ = "Merge Lists"
bUtil$ = "Utilities"
bPurge$ = "Purge Duplicates"
bOption$ = "Set Options"
bHelp$ = "Help"
bQuit$ = "Quit"
SET STATUS "Click, Type, or Select Menu Command"
activity$ = ""
DIRECTORY pth$
FORM 2
KEY 1,"!GOTO Helpme"
SET HEADING "Superbase Mailing List System"
MENU 1,0,1,"&Commands"
REM menu command selections will be assigned to variables clm% and
sel%

MENU ON clm%,sel%
ccr% = 0:cc% = 0:ent% = 0:a$ = "":ret% = 0:helpsel% = 0:zr% = 0
sel% = 0:a$ = ""
WAIT MENU OR KEY OR MOUSE
```

The user now has a choice of command strategies.

Most obviously, you can click the pushbutton that names the activity you want. Alternatively, you can choose a command from the pull-down Commands menu that replaces the normal Superbase menus. The third option is to type the initial letter of the pushbutton command you want to execute.

MAILIT.SBP detects the action taken and translates it into a value which then causes control to jump to the required subroutine. Here is the DML program code that accomplishes this:

MM1:

```

MENU 1,0,0," "
ON sel% GOTO AddEnts, MergeEnts, EditEnts, PrintLabs, Utilities,
PurgeEnts, SetOpts, Helpme, Quitit
REM ** If we get to here, the user must have pressed a key or clicked on
an area that is not a button
GET a$
GOSUB FlushBuffer
a$ = UCASE$ (a$)
SELECT CASE a$
  CASE "A"
    sel% = 1:REM was it the Add command?
  CASE "M"
    sel% = 2:REM Merge?
  CASE "E"
    sel% = 3:REM Edit?
  CASE "L"
    sel% = 4:REM Labels
  CASE "U"
    sel% = 5:REM Utilities
  CASE "P"
    sel% = 6:REM Purge
  CASE "O"
    sel% = 7:REM Options
  CASE "H", "?"
    sel% = 8:REM Help(!)
  CASE "Q", CHR$ (27)
    sel% = 9:REM Get outta dodge?
  CASE ELSE
    sel% = 0:REM Wasn't any key I know about, let's ignore it.
END SELECT
REM  A briefer alternative to the above CASE structure is:
REM  sel% = INSTR ("AMELUPOHQ",a$)
REM  if sel% is non-zero, then go find the appropriate subroutine
IF sel% THEN GOTO MM1: REM See above...
REM otherwise complain to user for mistyping, go back to the beginning
BELL : GOTO MainMenu

```

The code examples so far show how Superbase's DML combines BASIC-like commands with its own set of high level commands which achieve a great deal in a small space. The MENU and MENU ON commands, for example, economically display the user's preferred command text in a pull-down menu and detect any

selection made. The WAIT command waits for a user input through the specified channel, in this case menu, keyboard or mouse. ON ... GOTO is a familiar traditional BASIC command, whereas SELECT CASE is more likely to be found in modern variants of that language.

Mailit Activities

This section explains the core activities of the Mailit application. Each activity is performed by a routine in the MAILIT.SBP program, so after a description of the operation we provide an analyzed listing of the relevant sections of the code.

Maintaining the Mailing List

Adding new records to the mailing list is a primary activity, although in many situations lists are mainly managed by merging entire files. However, when the name and address data is captured on paper rather than supplied on disk, the data entry routine must be used.

Data Entry

AddEnts:

```
REM ** Add set up area
ret% = 9:helpsel% = 1:REM Codes so Help knows where to send us
back to
KEY 1,"!open form ~Mailit~:GOTO H1": REM set the Shift+F1 Help key
to the right page
SET HEADING "Add to Mailing List"
activity$ = "Data Entry Form":REM Identify the way the form is being
used
OPEN FORM "AddForm"
SET STATUS "ESC=Stop|SHIFT+TAB=Previous Field|F1=Help"
```

This first routine simply sets up user prompts and help, and opens the form needed for data entry. The next routine allows the user to enter record data.

AE1:

```
REM ** Actual guts of the Add routine.Checks for a blank form for an
end-of-entry
BLANK FORM :REM blank out a record so the user can enter into it
Country = "USA":REM preset the Country field for a US operation
VIEW :REM Show the new Country field.
REM Now enter the first screen of data, only the first 19 fields toREM
prevent carrying on to the next page
ENTER 1,19
REM if it's a blank form, go back to the main menu
```



```

IF FirstName = "" AND LastName = "" AND Company = "" THEN
activity$ = "":OPEN FORM "Mailit":GOTO MainMenu
STORE FORM :REM If it's not blank we store it
GOTO AE1 : REM Go back and enter another record

```

Once records exist, you must be able to change details and save them. The following routine (which does not actually follow on from the last one in the program code) is triggered by the Edit Mailing List pushbutton on the main menu screen. Again, the first subroutine sets up the screen, and the subsequent routines perform the actual editing, allowing you to browse through the file, modify records, or delete them.

Editing Records

EditEnts:

```

SET HEADING "Edit Mailing List"
ret% = 9:helpsel% = 3
KEY 1,"!open form ~Mailit~:GOTO H1"
KEY 2,"!GOTO Editit"
KEY 3,"!GOTO Deleteit"
KEY 10,"!GOTO DropFunctions"
activity$ = "Data Editing Form"
SELECT FIRST
PANEL ON
OPEN FORM "AddForm"
cc% = 1

```

EE1:

```

REM ** This actually searches for the keyboard input and turns the panel
REM on to allow the searching of records.
a$ = ""
SET STATUS "ESC=Stop|E=Edit|D=Delete|F1=Help"
WAIT PANEL OR KEY
GET a$
GOSUB FlushBuffer
a$ = UCASE$(a$)
IF a$ = "?" OR a$ = "H" THEN OPEN FORM "Mailit":GOTO H1
IF a$ = "E" THEN GOTO Editit
IF a$ = "D" THEN GOTO Deleteit
IF a$ = CHR$(27) OR a$ = "C" OR a$ = "Q" THEN GOTO
DropFunctions
GOTO EE1

```

Editit:

```

REM ** The user wants to edit this record, so we let them
SET STATUS "Editing Record..."
ENTER 1,19

```

```
STORE FILE "MailBase"  
GOTO EE1
```

Deleteit:

```
REM ** Are you sure you want to delete this record?  
SET STATUS "Delete Record..."  
REQUEST "Are you sure you wish to delete this record?", "", 130, n%  
IF n% THEN SELECT REMOVE FILE "MailBase"  
VIEW  
GOTO EE1
```

The next two routines have general applicability. Many programs need a way of clearing out function key assignments to prevent them interfering with other activities, and a "Press any key" routine has obvious uses. This is how to do it:

DropFunctions:

```
KEY 2  
KEY 3  
KEY 10  
cc% = 0  
PANEL OFF  
activity$ = ""  
OPEN FORM "Mailit"  
GOTO MainMenu
```

PressanyKey:

```
SET STATUS "Press any key to continue..."  
WAIT a$  
GOTO MainMenu
```

Merging and Purging

These activities are central to the mailing application. The program first decides whether the user has selected a merge or a purge, using the `prg%` variable to do so. It sets the opening screen accordingly:

MergeEnts:

```
REM ** Merge front end - The prg% variable is used to distinguish
whether we are merging (0), or purging (9)
FORM 6
SET STATUS "Merge Databases..."
ret% = 9:helpsel% = 2
KEY 1,"!GOTO H1"
merl% = 4:automan% = 1
SET HEADING "Merge Entries"
bHelp$ = "Help"
bCancel$ = "Cancel Merge"
bOK$ = "Begin Merge"
prg% = 0
GOTO MEBegin
```

PurgeEnts:

```
REM ** Purge front end
FORM 6
SET STATUS "Purge Databases..."
ret% = 9:helpsel% = 6
KEY 1,"!GOTO H1"
merl% = 1:automan% = 1
SET HEADING "Purge Entries"
bHelp$ = "Help"
bCancel$ = "Cancel Purge"
bOK$ = "Begin Purge"
prg% = 9
```

MEBegin:

```
SET STATUS "ESC=Stop|B=Begin|T=Set Thresholds"
cnt% = 0
GOTO ME21
```

At this point, control switches to the **me21** routine, described below, before returning to await user input on the form. The somewhat complex branching logic here needs careful attention.

ME1:

```
REM ** Wait for entry then see if it was a mouse click
VIEW
cnt% = 0:a$ = ""
WAIT KEY OR MOUSE
GET a$
GOSUB FlushBuffer
IF a$ <> "" THEN GOTO ME2
IF cnt% < 0 THEN GOTO H1
IF cnt% = 50 THEN GOTO MainMenu
IF cnt% = 999 THEN GOTO MChangePreset
IF cnt% = 0 THEN GOTO ME21
GOTO ME3
```

ME2:

```
REM ** Check which key was pressed -- typical use of SELECT CASE
z% = VAL (a$)
IF z% = 0 THEN z% = merl%
a$ = UCASE$ (a$)
SELECT CASE a$
CASE "H", "?"
GOTO H1
CASE "C", CHR$ (27)
GOTO MainMenu
CASE "V"
automan% = 2:GOTO ME21
CASE "M"
automan% = 1:GOTO ME21
CASE "O"
automan% = 3:GOTO ME21
CASE "T"
GOTO MChangePreset
CASE "B"
GOTO ME3
CASE ELSE
IF z% > 5 THEN GOTO ME1
merl% = z%
END SELECT
```

This routine, **me21**, sets up the duplicate detection thresholds, or levels, from values stored in the MDEFAULTS file, which is read in when MAILIT.SBP is first run.

ME21:

```

REM ** Set preset levels from pre-defined array
prs1% = prs%(merl%,1)
prs2% = prs%(merl%,2)
prs3% = prs%(merl%,3)
prs4% = prs%(merl%,4)
prs5% = prs%(merl%,5)
GOTO ME1

```

The following **MChangePreset** routine allows the user to vary the sensitivity of the duplicate checking procedure, then stores the new values in MDEFAULT.

MChangePreset:

```

REM ** Modify Preset levels and then store them
ENTER 1,5
prs%(merl%,1) = prs1%
prs%(merl%,2) = prs2%
prs%(merl%,3) = prs3%
prs%(merl%,4) = prs4%
prs%(merl%,5) = prs5%
OPEN "mdefault" FOR OUTPUT
? MEMORY
CLOSE OUTPUT
GOTO ME21

```

Duplicate Checking

The **me3** routine prepares for the Find Duplicates routine. It sets up the forms and then, if a merge is being performed, checks to see if the file is in a proper format and imports it.

ME3:

```

IF prg% = 0 THEN st$ = "Importing":st1$ = "Import" ELSE st$ = "Purging
Section of":st1$ = "Purge"
R% = RECCOUNT ("MailBase")
IF prg% = 0 THEN
  SET STATUS "Importing file..."
  REQUEST "File to import: ", "", 11,n%,fln$
  IF n% = 0 THEN GOTO MergeEnts
  IF EXISTS (fln$ + ".sbf") = 0 THEN REQUEST "Cannot find file ",
  fln$ + ".sbf",21,n%:GOTO MergeEnts
  bModify$ = "Importing...":bHelp$ = "Importing...":bCancel$ =
  "Importing...":bOK$ = "Importing..."
  VIEW
  RESTORE FieldNameData
  z1% = 0
  OPEN fln$ + ".sbd" FOR INPUT

```

```

I1:
  READ b$
  INPUT LINE a$
  IF RIGHT$ (a$,1) = ">" THEN z1% = 9
  a$ = TRIM$ ( LEFT$ (a$,15))
  IF UCASE$ (a$) <> UCASE$ (b$) THEN GOTO NoWay
  IF b$ = "Misc2" THEN GOTO OKDokey
  IF z1% THEN z1% = 0:INPUT LINE a$
  GOTO I1

NoWay:
  CLOSE INPUT
  REQUEST "This file does not match the MailBase", "parameters.
  Re-configure and try again.", 114,n%
  GOTO MainMenu

```

```

OKDokey:
  CLOSE INPUT
  IMPORT fln$ + ".sbf" TO FILE "MailBase"
  END IF

```

Once the import file has been merged into MAILBASE, the user is given the option to 'standardize' the records. This means using the conversion rules stored in RULEBASE to ensure that, for example, all occurrences of 'Ave' are changed to 'Avenue.' The chances of detecting duplicates are increased if the data has been standardized.

If the user does request standardization, the application switches to the **BegStand** routine, listed below, before resuming the duplicate checking procedure.

```

IF prg% = 0 THEN
  srtn% = 9
  REQUEST "Do You Wish to Standardize the Database First?", "", 130,n%
  IF n% = 0 THEN srtn% = 0:GOTO Beg
  GOSUB BegStand
  srtn% = 0
END IF

```

Following standardization, the **Beg** routine is executed. This in turn uses the core **FindDuples** routine before displaying the results of the operation and returning to the main menu.

Beg:

```

md% = RECCOUNT ("MailBase")
GOSUB FindDups
IF (prs1% + prs2% + prs3% + prs4% + prs5% = 0) AND prg% THEN
SELECT REMOVE FILE "MailBase"
fl% = RECCOUNT ("MailBase")
activity$ = st1$ + " complete" + CHR$ (13)
activity$ = activity$ + "Initial Record Count was " + LTRIM$ ( STR$
(R%,5,0)) + "," + CHR$ (13)
IF prg% = 0 THEN
    activity$ = activity$ + LTRIM$ ( STR$ (md% - R%,5,0)) + " Records
    Imported" + CHR$ (13)
END IF
activity$ = activity$ + LTRIM$ ( STR$ (md% - fl%,5,0)) + " Records
Purged" + CHR$ (13)
activity$ = activity$ + LTRIM$ ( STR$ (fl%,5,0)) + " Records Remaining"
+ CHR$ (13)
FORM 1
SET STATUS "Press any key to continue..."
WAIT KEY a$
GOTO MainMenu

```

The duplicate checking process makes use of some important features. After some preliminary checks it modifies the VirField field in MAILBASE, assigning it a new formula. The formula is derived from the thresholds set by the user earlier in the operation. It sets the number of characters from each threshold field that are to be used during the checking process. The more characters specified, the fewer the records that will be detected as candidates for deletion. For example, if you specified just one character in the last name, all records where the last name began with the same letter would match. However, if you specified six characters, only records where all six were the same would be picked up.

Once the program has modified VirField, the program builds an index using it. This index is the key to duplicate checking: *if any two index keys are the same, one of the two records must be a candidate for purging.*

FindDups:

```

IF prg% = 0 AND (prs1% + prs2% + prs3% + prs4% + prs5% = 0) THEN
RETURN
IF prg% AND (prs1% + prs2% + prs3% + prs4% + prs5% = 0) THEN
    REQUEST "WARNING! All thresholds are currently set to zero, this",
    "will delete all records in the file. Proceed?",143,n%
    IF n% = 0 THEN GOTO PurgeEnts
    REQUEST "Are you sure?",",",130,n%
    IF n% = 0 THEN GOTO PurgeEnts
END IF
SET STATUS "Removing current index..."

```

```

FILE "MailBase"
ON ERROR GOTO NoIndex
INDEX VirField.MailBase
REMOVE INDEX VirField.MailBase
GOTO GoOn

```

```

NoIndex:
RESUME GoOn
END

```

```

GoOn:
ON ERROR GOTO Errtrap
crit$ = "LEFT$(Zip.MailBase," + LTRIM$ ( STR$ (prs%(merl%,5),2,0)) +
") + LEFT$(lastname.Mailbase," + LTRIM$ ( STR$ (prs%(merl%,1), 2, 0))
+ ") + left$(Address1.MailBase," + LTRIM$ ( STR$ (prs%(merl%,2), 2, 0))
+ ") + left$(company.MailBase,"
crit$ = crit$ + LTRIM$ ( STR$ (prs%(merl%,3),2,0)) + " + left$(
City.MailBase, " + LTRIM$ ( STR$ (prs%(merl%,4),2,0)) + ")"
SET STATUS "Creating Index..."
bModify$ = "Wait..."
bHelp$ = "Wait..."
bCancel$ = "Wait..."
bOK$ = "Wait..."
FORM 6
MODIFY VirField.MailBase,"VirField;VTX CLC RDO;50;14;0",crit$
CREATE INDEX ON VirField.MailBase

```

At this point the index exists, and after some setting up the program can start looking for identical keys. To do this, it uses the SELECT DUPLICATE command.

```

IF automan% < 3 THEN
FORM 5
ELSE
activity$ = "Wait, " + st$ + " database...":FORM 1
END IF
SET STATUS st$ + " database..."
bKeep$ = "Keep":bDelete$ = "Delete":bRplace$ = "Replace"
IF automan% > 1 THEN bKeep$ = "Auto Mode":bDelete$ = "Auto
Mode": bRplace$ = "Auto Mode"
IF automan% = 2 THEN bDelete$ = "Searching..."
SELECT FIRST
recproc% = 0
numdups% = 0

```

These are the routines that run through every record in the database searching for duplicates. **FD1** sets up the 'Master' record against which any matching records will be compared:

FD1:

```
GET ink$
IF ink$ = CHR$ (27) OR ink$ = "C" THEN RETURN
a10$ = VirField.MailBase
recproc% = recproc% + 1
IF automan% < 3 THEN
    IF automan% = 2 THEN bDelete$ = "Searching..."
    a1$ = FirstName.MailBase
    a2$ = LastName.MailBase
    a3$ = Company.MailBase
    a4$ = Address1.MailBase
    a5$ = Address2.MailBase
    a6$ = City.MailBase
    a7$ = State.MailBase
    a8$ = Zip.MailBase
    a9$ = DATE$ (Entered.MailBase,"mm/dd/yyyy")
    a11$ = ""
    BLANK FORM :VIEW :SELECT CURRENT
END IF
```

The screen is set up, and we have our current index. Now, are there any duplicates?

FD2:

```
SELECT DUPLICATE
IF EOF ("MailBase") THEN
    SELECT NEXT FILE "MailBase"
    IF EOF ("MailBase") THEN RETURN
    GOTO FD1
ELSE
    IF automan% < 2 THEN
        numdups% = numdups% + 1
        BELL
        a11$ = VirField.MailBase
        VIEW
        bCmd% = 0
```

At this point the program has found a duplicate and, as the user has selected manual mode, we set up the display and present options for deleting either the duplicate or the original, or ignoring the match. If the variable automan% has the value 2, this means that the user has requested automatic processing, so all duplicates detected are removed without confirmation.

MF1:

```
SET STATUS "ESC=Stop|D=Delete|K=Keep|R=Replace"
WAIT KEY OR MOUSE
IF bCmd% = 0 THEN GET a$:a$ = UCASE$ (a$)
```

```

IF bCmd% = 3 OR a$ = "R" THEN a11$ = virfield.MailBase:SELECT
KEY a10$:SELECT REMOVE FILE "MailBase":SELECT KEY a11$:
GOTO FD1
IF bCmd% = 2 OR a$ = "D" THEN SELECT REMOVE FILE "MailBase":
SELECT KEY a10$:GOTO FD1
IF bCmd% = 1 OR a$ = "K" THEN
SELECT NEXT FILE "MailBase"
IF EOF ("MailBase") THEN RETURN
GOTO FD1
END IF
IF a$ = "C" OR a$ = CHR$ (27) THEN RETURN
GOTO MF1
ELSE
IF automan% = 2 THEN numdups% = numdups% + 1: bDelete$ =
"Deleting...": a11$ = VirField.MailBase:VIEW
SELECT REMOVE FILE "MailBase"
SELECT KEY a10$
GOTO FD1
END IF
END IF
GOTO FD1

```

Label Printing

Label printing in Mailit is selected with the Print Labels pushbutton on the Main Menu. It uses pre-existing labels definitions created in Superbase's native mode. These are SBB files, and the following routines use the REQUEST command to display a list of the SBB files in the current directory. The user simply selects the required labels format.

Note how the ON ERROR GOTO command is used in conjunction with the **CreateZip** routine to ensure that there is an index on the Zip field in the MAILBASE file. This is necessary because the LABELS command has no separate provision for sorting, and must use the currently selected index.

PrintLabs:

```

SET HEADING "Print Labels"
SET STATUS "Printing Labels..."
c$ = ""
fln$ = "*.SBB"
REQUEST "Select the Labels Definition", "To print with:", 18, n%, fln$
IF n% = 0 THEN GOTO MainMenu
ON ERROR GOTO CreateZip
INDEX Zip
GOTO Zon

```

```
CreateZip:
  CREATE INDEX ON Zip
  RESUME Zon
```

```
Zon:
  ON ERROR GOTO Errtrap
  LOAD LABELS fln$
  LABELS WHERE ASK
  CLOSE FIELDS
  GOTO MainMenu
```

Housekeeping Activities

Mailit includes housekeeping routines for Backup, Export and Restoring a MAILBASE file from a previously backed up file.

The Housekeeping section begins with the **Utilities** routine, which sets up the selection screen. **U1** then translates the user's selections into the appropriate action. This area of the application illustrates selection logic very clearly, showing how control can be switched from one input screen to powerful processing routines, which in turn switch back to the input screen until the user chooses to exit back to the main menu.

```
Utilities:
  SET HEADING "Mailing List Utilities"
  FORM 9
  SET STATUS "Select Utility..."
  KEY 1,"!goto H1"
  helpsel% = 5:ret% = 9
```

```
U1:
  a$ = "";utilsel% = 0
  WAIT KEY OR MOUSE
  GET a$:a$ = UCASE$(a$)
  GOSUB FlushBuffer
  SELECT CASE a$
    CASE "S"
      GOTO Convertit
    CASE "E"
      GOTO Exportit
    CASE "B"
      GOTO Backitup
    CASE "H","?"
      GOTO H1
    CASE "R"
      GOTO Restorebase
```

```

CASE "Q", CHR$ (27)
GOTO MainMenu
CASE ELSE
ON utilsel% GOTO Convertit, Exportit, Backupit, Restorebase,
MainMenu, H1
END SELECT
GOTO U1

```

Utilities gives access to three other routines: **Convertit**, **Exportit**, **Backupit**, and **Restorebase**.

The **Exportit** routine asks you to enter a name for the export file, invokes a filter dialog into which you can type search criteria, and then creates an ASCII file. Mailit already has values for field and record separators – these are held in the variables f1\$, f2\$, r1\$, and r2\$.

Exportit:

```

SET HEADING "Exporting Mail Base"
activity$ = ""
FORM 1
SET STATUS "Exporting..."
REQUEST "Name of Export File?", "", 4, n%, fln$, 50
IF n% = 0 THEN GOTO Utilities
activity$ = "Exporting Mail Base to File " + fln$ + "... "
VIEW
FILE "MailBase"
EXPORT TO fln$ WHERE ASK USING CHR$ ( VAL (f1$)) + CHR$ (
VAL (f2$)), CHR$ ( VAL (r1$)) + CHR$ ( VAL (r2$)), "0"
GOTO Utilities

```

The **Backupit** routine creates a new Superbase file from the current MAILBASE file, under a different name. You might wish to back up a MAILBASE file before restoring another previously backed up file to work with.

Backupit:

```

SET HEADING "Back Up Mail Base"
activity$ = ""
FORM 1
SET STATUS "Select File Name..."
REQUEST "Backup Database", "to Which File?", 11, n%, fln$
IF n% = 0 THEN GOTO Utilities
IF EXISTS (fln$) THEN OPEN FILE LEFT$ (fln$, LEN (fln$) -
4):REMOVE FILE LEFT$ (fln$, LEN (fln$) - 4)
activity$ = "Backing up Mail Base to the new Superbase file " +
fln$ + "... "
SET STATUS "Backing up your mail base..."
VIEW
FILE "MailBase"

```

```
SELECT ALL TO FILE fln$  
GOTO Utilities
```

The **Restorebase** routine first checks that the file the user specifies as a source to be made into a new MAILBASE has a compatible format. This is done in **Rb1** by checking the fieldnames in the SBD file of the named source against the field names in the **FieldNameData** routine later on in the program.

Restorebase:

```
SET HEADING "Restore Mail Base"  
SET STATUS "File to Back up from..."  
activity$ = ""  
FORM 1  
REQUEST "Restore from","Which File?",11,n%,fln$  
IF n% = 0 THEN GOTO Utilities  
IF EXISTS (fln$ + ".sbf") = 0 THEN REQUEST "Cannot find database " +  
fln$,"",113,n%:GOTO Utilities  
REQUEST "Proceeding with this action will overwrite","all data in the  
present database. Continue?",143,n%  
IF n% = 0 THEN GOTO Utilities  
RESTORE FieldNameData  
z1% = 0  
OPEN fln$ + ".sbd" FOR INPUT
```

Rb1:

```
READ b$  
INPUT LINE a$  
IF RIGHT$ (a$,1) = ">" THEN z1% = 9  
a$ = TRIM$ ( LEFT$ (a$,15))  
IF UCASE$ (a$) <> UCASE$ (b$) THEN GOTO GetoutofHere  
IF z1% THEN z1% = 0:INPUT LINE a$  
IF b$ = "Misc2" THEN GOTO Yeppers  
GOTO Rb1  
GetoutofHere:  
CLOSE INPUT  
REQUEST "This file is not in the Mail Base","format.Re-configure and try  
again.",113,n%  
GOTO Utilities
```

Once compatibility is confirmed, the following routine deletes the current MAILBASE, creates a new MAILBASE, and indexes it on the Lastname field.

Yeppers:

```
CLOSE INPUT  
activity$ = "Restoring database..."  
SET STATUS "Restoring database..."
```

```

FORM 1
cc% = 1
REMOVE FILE "MailBase"
OPEN FILE fln$
SELECT ALL TO FILE "MailBase"
CREATE INDEX ON lastname.MailBase
CLOSE ALL
cc% = 0
OPEN FORM "Mailit"
GOTO Utilities

```

Standardization

Convertit is designed to standardize MAILBASE by applying conversion rules taken from the RULEBASE file. As explained above, the purpose of standardization is to increase the chances of detecting duplicates in the file.

The logic in this part of the MAILIT.SBP program is complex. It is mainly concerned with checking every MAILBASE field for which there is an entry in RULEBASE, and seeing whether it contains anything that needs to be altered. If it does, **Convertit** or one of its subordinate routines makes the appropriate change and saves the new details. This is repeated for each record in MAILBASE, so the process can be lengthy.

BegStand:

```

activity$ = ""
FORM 1
GOTO CI1

```

Convertit:

```

activity$ = ""
FORM 9
SET STATUS "Are you sure?"
SET HEADING "Standardize Mail Database"
OPEN FILE "RuleBase"
REQUEST "This process can take a considerable time for " + LTRIM$ (
STR$ ( RECCOUNT ("MailBase"),5,0)) + " records," "do you wish to
continue?",143,n%
IF n% = 0 THEN GOTO Utilities

```

CI1:

```

SET STATUS "Standardizing Mail Base..."
activity$ = "Processing Rule Base..."
FORM 1
ctx% = 1
UPDATE RuleNum.RuleBase = ctx%:ctx% = ctx% + 1
ctx% = 1

```

```

UPDATE RecNum.MailBase = ctx%:ctx% = ctx% + 1
INDEX RuleNum.RuleBase
FORM 10
rctotal% = RECCOUNT ("MailBase")
rcleft% = RECCOUNT ("MailBase")
rcnum% = 0
totrul% = RECCOUNT ("RuleBase")
onrul% = 0
GOSUB UpdateScreen
FILE "MailBase"
INDEX recnum.mailbase
SELECT FIRST FILE "MailBase"
SELECT FIRST FILE "RuleBase"

```

In the following C1 routine, the program builds the string zz\$, which contains what is actually being looked for. The variable nsp% may be set to 0 to signify that a space at the end is not important, or 1 to signify that it is important.

The variable wh% is set to 1 if zz\$ can occur anywhere in the field, 2 if it can only be at the end, or 3 if only at the beginning.

C1:

```

GET k$
IF k$ = CHR$ (27) THEN SELECT WHERE :SELECT FIRST :GOTO
EndStand
IF EOF ("RuleBase") THEN GOTO Cend
onrul% = onrul% + 1
rcnum% = 0
rcleft% = RECCOUNT ("MailBase")
GOSUB UpdateScreen
nsp% = 0:wh% = 1
z$ = LookFor
IF LEFT$ (z$,1) = "*" THEN z$ = MID$ (z$,2):wh% = 2
IF LEFT$ (z$,1) = "^" THEN z$ = MID$ (z$,2):wh% = 3
zz$ = ""
FOR x% = 1 TO LEN (z$)
  IF MID$ (z$,x%,1) = "!" THEN
    zz$ = zz$ + ""
  ELSE IF MID$ (z$,x%,1) = "@" THEN
    zz$ = zz$ + " "
  ELSE
    zz$ = zz$ + MID$ (z$,x%,1)
  END IF
NEXT x%
IF INSTR (z$,"!") THEN nsp% = 1
ex$ = "select where " + FieldName.RuleBase + " like " + CHR$ (34) + ""
+ LTRIM$ ( TRIM$ (zz$)) + "" + CHR$ (34)

```

```
EXECUTE ex$
ex$ = ""
SELECT FIRST FILE "Mailbase"
```

C20:

```
ex$ = "a$" + FieldName.RuleBase
EXECUTE ex$
ex$ = ""
a$ = a$ + " "
IF EOF ("MailBase") THEN SELECT NEXT FILE "RuleBase":GOTO C1
rcnum% = Recnum.mailBase
rcleft% = rcotal% - rcnum%
GOSUB UpdateScreen
```

CRemake:

```
REM ** Is zz$ in a$? If yes, then we possibly have a match
zz% = INSTR ( UCASE$ (a$), UCASE$ (zz$))
IF zz% = 0 THEN SELECT NEXT FILE "MailBase":GOTO C20
IF nsp% = 1 AND MID$ (a$,zz% + LEN (zz$),1) <> " " THEN SELECT
NEXT FILE "MailBase":GOTO C20
IF zz% <> 1 AND wh% = 3 THEN SELECT NEXT FILE
"MailBase":GOTO C20
IF wh% = 2 THEN
    IF zz% + LEN (zz$) + 4 < LEN (a$) THEN
        SELECT NEXT FILE "MailBase":GOTO C20
    END IF
END IF
IF zz% < 2 THEN ff$ = "" ELSE ff$ = LEFT$ (a$,zz% - 1)
af$ = MID$ (a$,zz% + LEN (zz$))
Ct$ = ChangeTo
pw% = 1
IF LEFT$ (Ct$,1) = "*" THEN pw% = 2:Ct$ = MID$ (Ct$,2)
IF LEFT$ (Ct$,1) = "^" THEN pw% = 3:Ct$ = MID$ (Ct$,2)
ch$ = ""
FOR x% = 1 TO LEN (Ct$)
    IF MID$ (Ct$,x%,1) = "@" THEN ch$ = ch$ + " " ELSE ch$ = ch$ + MID$
(Ct$,x%,1)
NEXT x%
IF UCASE$ ( MID$ (a$,zz%, LEN (ch$))) = UCASE$ (ch$) THEN
SELECT NEXT FILE "MailBase":GOTO C20
IF INSTR (Ct$,"~") THEN ch$ = " "
IF pw% = 1 THEN a$ = ff$ + ch$ + af$
IF pw% = 2 THEN a$ = ff$ + af$ + ch$
IF pw% = 3 THEN a$ = ch$ + ff$ + af$
ex$ = FieldName.RuleBase + "a$"
```



```
a$ = LTRIM$ ( TRIM$ (a$))
EXECUTE ex$
ex$ = ""
STORE FILE "MailBase"
SELECT NEXT FILE "MailBase"
GOTO C20
```

Cend:

```
CLOSE ALL
activity$ = "Standardization Complete." + CHR$ (13)
IF srtn% THEN activity$ = activity$ + "Now Processing Merge File..." +
CHR$ (13)
OPEN FORM "Mailit"
WAIT FOR 2
```

EndStand:

```
IF srtn% THEN RETURN
GOTO Utilities
```

UpdateScreen:

```
REM ** this updates the screen so the user can see why we are taking
fourteen hours away from their favorite mailbase file
onrul$ = LTRIM$ ( STR$ (onrul%,5,0)) + " of " + LTRIM$ ( STR$
(totrul%,5,0)) + "."
rcleft$ = LTRIM$ ( STR$ (rcleft%,6,0))
rcnum$ = LTRIM$ ( STR$ (rcnum%,6,0))
IF rcnum% = 0 THEN rcnum$ = "Searching..."
VIEW
RETURN
```

Maintaining the RULEBASE

The next major section in MAILIT.SBP is concerned with maintaining the RULEBASE file. The user is allowed to add, change or delete rules, using the notation described in the **Optsrule** routine. The section begins with the usual screen setup and user input detection routines:

SetOpts:

```
SET HEADING "Set Mail Options"
bCancel$ = "Cancel"
FORM 7
SET STATUS "ESC=Stop|M=Modify Rules|C=Change Separators"
MENU 1,0,0," "
SO1:
optsel% = 0:a$ = ""
```

```

WAIT KEY OR MOUSE
IF optsel% = 0 THEN GET a$:a$ = UCASE$ (a$):GOSUB FlushBuffer
IF a$ = "M" OR optsel% = 1 THEN GOTO OptsRule
IF a$ = "C" OR optsel% = 2 THEN GOTO OptsSep
IF a$ = CHR$ (27) OR a$ = "Q" OR optsel% = 4 THEN GOTO MainMenu
IF a$ = "H" OR optsel% = 5 THEN helpsel% = 7:ret% = 9:GOTO H1
GOTO SO1

```

OptsRule:

```

REM Here we change and/or add all the rules for the rule base.
REM The rules are:   WHEN SEARCHING:
REM * - Look for this string at the end of the field
REM ^ - Look for this string at the beginning of the field
REM @ - this is the same as a space
REM ! - do not allow the search field to end in a space
REM   WHEN REPLACING:
REM * - Put the resultant string at the end of the field
REM ^ - Put the resultant string at the start of the field
REM @ - Just a space
REM ~ - Delete this entirely (put nothing in its place)
SET HEADING "Modify Rule Base"
SET STATUS "Select field to modify..."
FILE "MailBase"
REQUEST "Select Field","",9,n%,flnm$
OPEN FILE "RuleBase"
IF n% = 0 THEN GOTO SetOpts
ctx% = 1
flnm$ = TRIM$ ( LEFT$ (flnm$,15))
UPDATE RuleNum.RuleBase = ctx%:ctx% = ctx% + 1 WHERE
FieldName.RuleBase = flnm$
FILE "RuleBase"
ex$ = "select where FieldName.RuleBase LIKE " + CHR$ (34) + flnm$ +
CHR$ (34)
EXECUTE ex$
ex$ = ""
PANEL ON
OPEN FORM "optsel2"
mdf$ = "Modifying the " + flnm$ + " field"
cc% = 1
SELECT FIRST

```

Op:

```

REM ** this area allows entry into the rule base (and editing, deleting,
etc.)
SET STATUS "ESC=Stop|N=New|D=Delete|E=Edit|M=Modify Other"

```

```
a$ = ""
WAIT PANEL OR KEY
SET STATUS "ESC=Stop|N=New|D=Delete|E=Edit|M=Modify Other"
GET a$
GOSUB FlushBuffer
a$ = UCASE$(a$)
IF a$ = "N" THEN GOTO Op1
IF a$ = "M" THEN CLOSE FORM :GOTO OptsRule
IF a$ = "D" THEN GOTO Op2
IF a$ = "E" THEN GOTO Op3
IF a$ = CHR$(27) OR a$ = "C" THEN cc% = 0:PANEL OFF :SELECT
WHERE :OPEN FORM "Mailit":GOTO MainMenu
GOTO Op
```

Op1:

```
REM ** Add new rule
n$ = "":ct$ = ""
REQUEST "Enter Value to Look for in Field " + flnm$ + "?", "", 4, n%, n$, 15
IF n% = 0 THEN GOTO Op
BLANK FILE "RuleBase"
FieldName.RuleBase = flnm$
LookFor.RuleBase = n$
REQUEST "Change " + n$ + " to what?", "", 4, n%, ct$, 40
IF n% = 0 THEN GOTO Op
ChangeTo.RuleBase = ct$
RuleNum.RuleBase = ctx%
ctx% = ctx% + 1
STORE FILE "RuleBase"
GOTO Op
```

Op2:

```
REM ** Delete a rule
REQUEST "Delete the Change " + LookFor + " to", ChangeTo + "
record?", 130, n%
IF n% = 0 THEN GOTO Op
SELECT REMOVE FILE "RuleBase"
FORM 1
GOTO Op
```

Op3:

```
REM ** Edit a rule
REQUEST "Change the " + LookFor + " to " +
ChangeTo, "Record?", 130, n%
IF n% = 0 THEN GOTO Op
n$ = LookFor.RuleBase
```

```

REQUEST "Alter Look For to ", "", 4, n%, n$, 15
IF n% = 0 THEN GOTO Op
LookFor = n$
n$ = ChangeTo
REQUEST "Alter Change To Field to ", "", 4, n%, n$, 40
IF n% = 0 THEN GOTO Op
ChangeTo = n$
STORE FILE "RuleBase"
VIEW
GOTO Op

```

The other choice that the user has on the Options screen is to alter the field and record separators used in the **exportit** routine.

OptsSep:

```

REM ** Change the field and record separators
bCancel$ = "Cancel"
fo1$ = f1$:fo2$ = f2$:ro1$ = r1$:ro2$ = r2$
SET HEADING "Change Export Separators"
FORM 8
SET STATUS "ESC=Stop|F=Fields|R=Records"

```

OS1:

```

optsel% = 0:a$ = ""
WAIT KEY OR MOUSE
IF optsel% = 0 THEN GET a$:a$ = UCASE$(a$):GOSUB FlushBuffer
IF a$ = CHR$(27) OR a$ = "C" OR optsel% = 99 THEN f1$ = fo1$:f2$ =
fo2$:r1$ = ro1$:r2$ = ro2$:GOTO MainMenu
IF a$ = "U" OR optsel% = - 999 THEN GOTO MainMenu
IF a$ = "F" THEN ENTER 6,2:GOTO Checkup
IF a$ = "R" THEN ENTER 8,2:GOTO Checkup
GOTO OS1

```

Here we check to make sure the user typed in values that are valid ASCII field delimiters. If the values are not allowed, we change them back to what they were before.

Checkup:

```

IF VAL (f1$) = 0 AND ASC (f1$) <> 48 THEN f1$ = fo1$
IF VAL (f2$) = 0 AND ASC (f2$) <> 48 THEN f2$ = fo2$
IF VAL (r1$) = 0 AND ASC (r1$) <> 48 THEN r1$ = ro1$
IF VAL (r2$) = 0 AND ASC (r2$) <> 48 THEN r2$ = ro2$
IF VAL (f1$) < 0 OR VAL (f1$) > 255 THEN f1$ = fo1$
IF VAL (f2$) < 0 OR VAL (f2$) > 255 THEN f2$ = fo2$
IF VAL (r1$) < 0 OR VAL (r1$) > 255 THEN r1$ = ro1$
IF VAL (r2$) < 0 OR VAL (r2$) > 255 THEN r2$ = ro2$
VIEW
GOTO OS1

```

The Mailit Help System

Mailit includes a simple help file comprising a key field and a long text field. The key provides a way of linking each record in the help file to a specific part of the application. The text fields explain the various aspects of the system.

When the user requests help from the main menu, Mailit opens the help topic selection screen and displays it. The user may then pick any help topic.

The HELPBASE file itself is opened either when the user has chosen a help topic, or when Help is selected from another part of the application.

Example Code: Help

Helpme:

```
REM ** Main entrance to the Help area
SET HEADING "MailBase Help"
ret% = 0:helpsel% = 0
```

HON:

```
REM ** All the help calls from utilities come in here
FORM 3
SET STATUS "Select Help Topic"
MENU OFF
IF ret% THEN GOTO H1
```

HNK:

```
REM ** help main menu
WAIT KEY OR MOUSE
GET a$:hl% = INSTR ("AMELUPOQS", UCASE$(a$))
GOSUB FlushBuffer
IF a$ = CHR$(27) OR hl% = 8 OR helpsel% = 8 THEN GOTO EndHelp
IF helpsel% + hl% = 0 THEN GOTO HNK
IF helpsel% = 0 THEN helpsel% = hl%
```

H1:

```
REM ** actual help area.displays proper screen
OPEN FILE "HelpBase"
hl$ = "<Esc> - Quit back to Main | Any other key for more Help."
IF ret% THEN hl$ = "<Esc> - Quit back to Start | Any other key for more Help."
SET STATUS hl$
SELECT WHERE Topic = helpsel%
SELECT FIRST
FORM 4
IF zr% = 0 AND ret% THEN zr% = helpsel%
WAIT KEY OR MOUSE
```

```

GET a$
GOSUB FlushBuffer
IF a$ = CHR$ (27) OR UCASE$ (a$) = "Q" THEN GOTO EndHelp
GOTO Helpme
EndHelp:
REM ** return to the proper area
ON zr% GOTO
AddEnts, MergeEnts, EditEnts, PrintLabs, Utilities, PurgeEnts, SetOpts
GOTO MainMenu

```

Exit and General Purpose Routines

Last in the program code are the exit routine and three others which do not need to be in line with other code sections.

Quitit asks the user to confirm the exit action, then saves the contents of memory to the MDEFAULT file, ready for reloading at the start of the next session.

Quitit:

```

REM ** quit routine
REQUEST "Confirm Exit...", "", 143, n%
IF n% = 0 THEN GOTO MainMenu
OPEN "MDefault" FOR OUTPUT
? MEMORY
CLOSE OUTPUT
CLOSE ALL
SET HEADING ""
END

```

FieldNameData and **FlushBuffer** are invoked from other areas of the program.

FieldNameData:

```

REM this is the data used to make sure the database is in the same
format as the mailbase file
DATA "RecNum", "VirField", "Title", "FirstName", "LastName",
"Company", "Address1", "Address2", "City", "State", "Zip", "Country",
"HomePhone", "WorkPhone", "FAX", "Notes", "Entered", "Misc1", "Misc2"

```

FlushBuffer:

```

REM get rid of all those extra typed characters
GET zz1$
IF zz1$ = "" THEN RETURN
GOTO FlushBuffer

```

The MAILIT error handling routine appears as the last section of the program.

Errtrap:

```
REM an error has occurred.The actual error is in ERRNO, and the name
of the error can be acquired with the ERR$() command
REM an error number 11 is a control-c or a Stop button
REM a RESUME is necessary to clear the Error condition and to resume
with the normal program execution
REM ** cc%=0 --- error occurred with program.Non-expected.
REM ** cc%=1 --- Expected error, return back to next statement (stay in
same routine) -- I use this for the end-of-file error that occurs when you
use WAIT PANEL
IF ERRNO = 10 AND cc% = 1 THEN SET STATUS "*** End of File
***":BELL :WAIT FOR .5:RESUME NEXT
IF cc% = 1 AND ERRNO = 11 THEN cc% = 0:DIRECTORY pth$:OPEN
FORM "Mailit":RESUME MainMenu
IF ERRNO = 11 THEN DIRECTORY pth$:OPEN FORM "Mailit":
RESUME MainMenu
IF cc% = 1 THEN RESUME NEXT
SET STATUS "*****" + ERR$ ( ERRNO ) + "*****"
WAIT FOR 1.2
DIRECTORY pth$
OPEN FORM "Mailit"
RESUME MainMenu
```

—

—

—

22 THE TRADING SYSTEM

The Trading system aims to fulfil a wider, more diverse set of requirements than the previous example. This makes it the most realistic prototype of a full-scale commercial application.

The Trading system designer has analyzed the various computing needs of a hypothetical international securities trading firm, ranging from the production of client lists, through the calculation of the dollar amount due on each transaction, to the mailing of contract advices and statements to clients.

The designer has then used Superbase to create a single, cohesive system framework within which the diverse application requirements are met using an interrelated set of file definitions, programs, forms and text documents.

This chapter is aimed at those developers planning to implement a full-scale system. It accordingly reflects the higher level of complexity inherent in professional applications of this type.

The best way of getting started with the Trading system is to try out some of the functions for yourself. The system is 'sealed,' in the sense that all functions are under program control. You can access the application simply by selecting DML Run and running the MENU program.

At the top level the application uses the command strategy of pull-down menus. These menus are defined in and displayed by MENU. Some of the program routines corresponding to individual menu options are contained within MENU, while others are structured in separate program modules. These modules are called (chained) by MENU. These modules always chain back to MENU on completion.

Many of the menu options open forms. Once a form is open, the system deploys a different command strategy: the forms themselves include pushbuttons for activities like entering a new record, editing an existing record, browsing and so on.

It is anticipated that users designing applications of the complexity of the Trading System will own systems capable of displaying in high resolution without flicker. The form designs accordingly take advantage of 'hi-res' to include additional detail.

Forms are included that illustrate the use of radio buttons and check boxes as a more graphical means of displaying data. Some forms display details from a single record on multiple pages, while others are designed to display details from a number of records on a single page, using the Form Designer's one-to-many replication feature.

The Trading system has three main files: a client file, a stock file and a transaction file. The client file refers to a table of countries, and the stock file to a table of currencies. The transaction file includes LOOKUPS to both client and stock files. .

During transaction entry, data held in the stock, currency and client files is used in deriving calculated values for the transaction. These calculations are specified as part of the underlying transaction file definition itself. Referential integrity is achieved by including LOOKUP validations as part of the file definitions, and also by including programmed checks prior to the amendment of keys or the deletion of records in subsidiary files.

Color is used as an aid to identifying the file being manipulated. Data from the client file is displayed in yellow, data from the stock file in blue, and so on. Functions are provided which use the SELECT command (similar to the Process Query menu option) to create temporary Superbase files containing selected details from a number of other files. These temporary files are used as a basis for mailmerge operations. Following printing, there are examples of the use of global updating to update printing flags. Finally, as a means of further controlling this aspect of the application flow, certain menu options are grayed or re-activated following the completion of the relevant tasks.

The Query menu options show how it is possible to provide users with an ad hoc querying facility under program control. A number of reports are also included. The report formats were defined using Superbase's Report Generator, and the resulting DML report programs edited to bring them into line with the rest of the system.

The Trading system exemplifies a wide range of Superbase facilities and techniques, all within a structured framework. The following sections cover the main elements of the system – file definitions, programs and forms – in some detail.

Superbase is a very flexible system for application development. Processing rules, for example, can be included in forms, in programs, or right at the data definition level. The benefit of including rules with data definitions is that they are then defined once for the entire application, and are automatically applied in all circumstances. This simplifies the design and coding of other parts of the system. In the Trading system, validation and calculation rules are included wherever possible at this level. As a result, the file definitions may at first sight appear rather complex. On a first reading, the best approach might be to skim over these, then read them again in conjunction with the Activities sections.

Purpose of the Application

The Trading company trades in stocks quoted on various exchanges and in various currencies. The firm buys and sells stocks on behalf of and according to instructions given by its clients in various countries. The purpose of the application is to automate as far as possible the clerical and administrative aspects of this trading activity.

Overview of Activity

In any enterprise, certain procedures are one-time only and others must be carried out on a regular basis each day, each week and so on.

The initial set-up activities required before the application can be used on a day-to-day basis might be as follows:

- Enter new records for each country and currency
- Enter a new record for each existing client
- Enter a new record for each traded stock
- Print a Client Telephone List
- Print a Traded Stocks Report

The daily cycle of activities that comprise the work flow for the hypothetical trading firm are described below. In a real-world situation, there would be additional cycles of periodic and annual activity.

- Enter today's exchange rates and stock prices
- Enter new stock information from analysts' reports
- Print and circulate the Traded Stocks and the New Highs and New Lows reports. These provide sales people with a summary of pricing and other useful stock data
- Run the New Highs and New Lows updates to record new price highs and lows in the stocks file
- Telephone clients, entering sales or purchases as they are made, and updating client information
- Enter certificate numbers of new stock certificates received
- Enter cash receipts from clients
- Print personalized Certificate Advices
- Update transactions where a Certificate has been produced
- Print personalized Contract Notes
- Update transactions where a Contract Note has been produced
- Print and circulate the Trading Balance reports
- Telephone selected clients concerning the status of their account
- Back up all data files

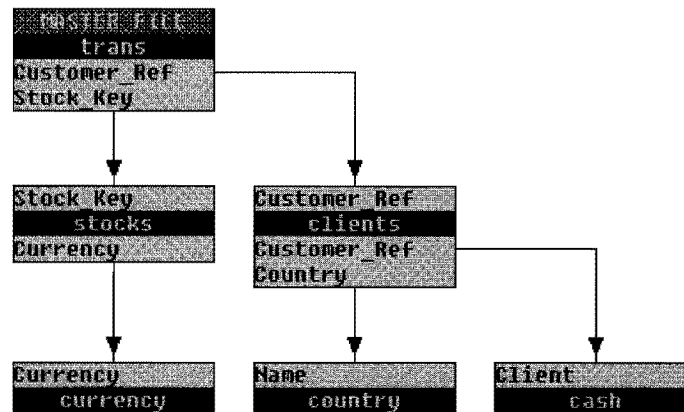
Understanding the work flow is critical to the success of an application. Files must be kept up to date, documents produced and circulated, and routine 'housekeeping' procedures carried out on time and in the right sequence.

A well-designed system will take account of all these factors.

The Database Structure

The Trading System database is a multi-file structure: there are three main files, CLIENTS, STOCKS and TRANS, and a number of ancillary files, including the COUNTRY and CURRENCY look-up files, the CASH postings file, and a control file CTRL. The control file contains a number of flags to ensure that certain routines are performed in the correct sequence. The central file in the system is the TRANS transaction file, in which details are recorded of each stock sale or purchase. Since the firm repeatedly trades with the same clients and in the same stocks, client and stock details are stored in separate files. These details can then be accessed by code rather than re-entered for each transaction.

The client and stock information is accessed from the TRANS file by means of LOOKUPS or links using the Customer_Ref and the Stock_Key as uniquely identifying codes or keys. Similarly, since many clients may live in the same country, and many stocks are quoted in the same currency, common country and currency details are also held in separate files.



The above figure shows the dependency relationships between the major files.

The Menu Structure

Access to the application is controlled through a menu which organizes the various activities into groups.

File		Daily		Documents	
Clients	AC	Currency Rates	A	Certificates	A
Stocks	AS	Stock Prices	A	Contract Notes	A
Countries	AO	Certificate Nos	AN	Statements	A
Currencies	AU	Cash Receipts	AR		
Transactions	AT	Cash Payments	AP		
Exit	AE				

Update		Query		Reports	
Stock Highs	AH	Run	A	Traded Stocks	A
Stock Lows	AL	New	A	Stock Highs	A
Certs Sent	A	Open	A	Stock Lows	A
Notes Sent	A	Edit	A	Trading Balances	A
Reset	A	Save	A	-----	
				Telephone Nos	A
				Client Balances	AB
				Cash Audit	A

The File Menu

This group consists of the maintenance and enquiry activities for the main files in the application. Each of the main files CLIENTS, STOCKS, TRANS, CURRENCY and COUNTRY has an associated entry on this menu.

The individual activities of data entry, editing and deletion are accessed from pushbuttons on the forms themselves. There are also pushbuttons for changing the browsing sequence and for activating the browsing panel under program control.

The maintenance form for transactions includes pushbuttons for viewing full details of the relevant client or stock. The clients form includes a page showing the transactions for each client. You can browse through client transactions, a page at a time, then decide to view the maintenance form for an individual transaction or stock record. You simply click on the Trans or Stocks pushbutton and then click on the relevant transaction line. The maintenance form for stocks also includes a similar page showing the transactions for each stock, with facilities to view full details for a transaction or client. This is an example of the flexibility that is possible in a Superbase application.

The Daily Menu

This group consists of daily activities including entry of the latest currency rates, stock prices, certificate numbers, cash receipts and payments. These routines are organized under a common heading to highlight the fact that they are part of the daily cycle. Some of these routines are "streamlined" versions of the editing routines provided through the File menu. Daily Certificate Nos, for example, displays a number of selected transactions for editing on a single screen page.

The Documents Menu

This group consists of the document production routines. One of the main benefits of a computerized system is the automatic, timely and accurate production of routine documents recording details of business transactions. This system produces personalized contract notes for each transaction, advice notes to be sent out with each stock certificate received on behalf of a client, and statements showing the transaction details and the balance owing to or due from each client.

The Update Menu

This group consists of updating routines which must be carried out from time to time on various files. The current year's high and low prices and price dates are periodically updated. Also, flags in the transaction file must be updated once the relevant documents have been produced. Following document production, the system prevents new transactions from being entered by graying the appropriate menu option until the updates have been carried out.

The Query Menu

This menu enables users to create new queries on the database and to load, edit or run existing queries. It employs the Query dialog under program control, enabling the user to specify ad hoc reports interactively without having to resort to the Report Generator or programming language.

The Reports Menu

This final group consists of the programmed reports generated for the Trading system. These include trading balances by stock and by client. The report programs were produced by the Report Generator, then edited to include more comprehensive error handling, a repeat print option, a return to the MENU program and other features.

Filenaming Conventions

The Trading system uses many files, programs and forms. Naming conventions for these items have accordingly been adopted to assist in identifying their functions. Following Windows conventions, each menu option has a keyboard letter equivalent. This is underlined on-screen, and indicated in the menu program command by an "&" preceding the letter, for example:

MENU 1,1,1,"&Clients"

Each item within a menu column has a unique letter associated with it. Hence the following entry has the letter "u" marked rather than the letter "C":

MENU 1,4,1,"&Currencies"

The conventions are as follows:

- Names of data files are purely descriptive. Examples: STOCKS, CURRENCY
- The master menu program is MENU and banner form MENUF
- All other files start with the letters STK
- All other programs chained to by the master menu program have the letters STK followed by the two keyboard equivalent letters of the menu and the item.

Examples: STKDC Daily Cash Rceipts
 STKDC Daily Currency Rates

- Forms associated with the File menu options have the letters STK followed by a single keyboard equivalent letter for the menu item. Examples: STKCFile Clients

 STKO File Countries

- Forms associated with other menu options have the letters STK followed by the same two letters as the programs that use them.

Examples: STKDR Daily Cash Rceipts
 STKOS Documents Statements

- Printed forms are as above but followed by a 2.

Examples: STKC2 File Clients

- Forms specific to EGA displays are as above but followed by an E.

Example: STKOS2E Documents Statements printed EGA version

In form design, colors are used as an aid to identifying files, as follows:

Clients:	yellow
Stocks:	light blue
Transactions:	green
Currency:	dark blue
Country:	red.

Files

The CLIENTS File

The CLIENTS file contains details of clients on whose behalf the trading company buys and sells stocks. The CLIENTS file maintenance function is accessed from the File menu, and can also be accessed from the STOCKS or TRANS maintenance forms. The MENU program includes a section labeled **stkf** and containing the routines responsible for opening the relevant forms and carrying out the maintenance functions for the CLIENTS, STOCKS and TRANS files. The CLIENTS display form is STKC and the print form STKC2. Selected CLIENTS information together with all related TRANS records may be viewed by accessing Statements from the Documents menu. The **stkf** section of MENU is also responsible for this function. The Statements display form is STKOS and the print form STKOS2.

Included in the CLIENTS file is a Country field. When creating or editing the Country field, Superbase performs a LOOKUP to a separate COUNTRY file. This is an example of cross-file validation. The instruction for Superbase to do this is contained within the CLIENTS file definition, and ensures that every CLIENTS record contains a valid key for Country. Prior to amendment or deletion of COUNTRY records, a check is programmed to ensure that no related CLIENTS records exist (see STKFO). Combined with the use of LOOKUP during creation, this check ensures the integrity of the reference from CLIENTS to COUNTRY files.

CLIENTS File Definition

Title	TXT	6 C
Firstname	TXT	18 C
Initials	TXT	5 U
Lastname	TXT IXD	18
Customer_Ref	TXT CLC RDQ IXD	8
LEFT\$ (UCASE\$ (Lastname.CLIENTS) + " ",4) + STR\$ (Serial.CLIENTS,"0000.")		

This read-only calculation creates a unique key for the client record by taking the first 4 characters of the Lastname, filling up with spaces, then adding a four character string padded with leading zeros. This string is made by converting the unique number held in the Serial.CLIENTS field.

Serial.CLIENTS (see below) is defined as SER ("CLIENTS"). The SER function yields a unique number for the current file CLIENTS, like a record number. The Customer_Ref is the link between the TRANS file and the CLIENTS file and is referred to by a LOOKUP in the TRANS file. Prior to deletion of a CLIENTS record, a programmed check is carried out in the **stkf** routine to ensure that no related TRANS records exist.

Class TXT VAL 1 U
 Class.CLIENTS = "B" OR Class.CLIENTS = "P"

Class.CLIENTS includes a validation formula to ensure that its value is always entered as "B" for Business or "P" for Private.

Company TXT 30
 Trading_Balance NUM IXD -,9999999.00

This balance is automatically maintained during the creation, amendment or deletion of related TRANS records by the MENU routine **stkf**.

Cash_Receipts NUM -,9999999.00

This balance is automatically increased by the value of cash receipts and reduced by the value of cash payments entered daily through the STKDR program.

Net_Due NUM CLC RDO -,9999999.00
 Trading_Balance.CLIENTS - Cash_Receipts.CLIENTS

The Net Due is automatically calculated by subtracting the total cash receipts from the total trading balance.

Telephone TXT 16
 Street TXT 30
 Address TXT 30
 City TXT IXD 18
 State TXT 18
 Zip_Code TXT 10
 Country TXT VAL IXD 24

LOOKUP (Country.CLIENTS,Name.COUNTRY) ELSE REQUEST "Country not found", "Select another country", 20,, Country.CLIENTS, 24, Name.COUNTRY

In the sample application there is a separate COUNTRY file, containing country details. On creation of a new CLIENTS record, this validation performs a LOOKUP, attempting to find a match between the value entered for Country.CLIENTS and Name.COUNTRY, the equivalent indexed field in the COUNTRY file. If no match is found, a type 20 dialog is displayed to force entry of a valid country name.

Start_Date DAT mmm dd,yy
 Last_Contact DAT mmm dd,yy
 Result TXT 60
 Next_Contact DAT mmm dd,yy
 Subject TXT 60
 Notes EXT 20

The Notes.CLIENTS field is an external field, designed to store the name of a Superbase SBT text file. In the application, related text files are held in the same

directory as the CLIENTS file, and by convention have names of the form Customer_Ref.SBT. Notes.CLIENTS is displayed on the second data page of the STKC form.

Serial NML CON RDO 999999.
SER("CLIENTS")

Serial.CLIENTS is a long integer set to the constant value SER("CLIENTS"). This function behaves much like a record number. As a read-only constant, its value is calculated once at the commencement of data entry and cannot be modified. Otherwise a change in its value would trigger a change in the value of Customer_Ref, thereby destroying the link between TRANS and CLIENTS.

Client VTX CLC RDO 30
Title.CLIENTS + " " + Firstname.CLIENTS + " " + Lastname.CLIENTS

Client.CLIENTS is a virtual calculation that concatenates the title, firstname and lastname fields. Storing these items in separate fields is useful for mailmerge purposes, so that letters can begin, for example, "Dear Mr Cellers", or even "Dear William". However, for other purposes it may be more convenient to use the concatenation "Mr William Cellers", rather than having to manipulate the fields separately.

Location VTX CLC RDO 30
City.CLIENTS + ", " + State.CLIENTS + " " + Country.CLIENTS

Location.CLIENTS is a virtual calculation that concatenates the city, state and country fields. This is used along as an abbreviated location with Client.CLIENTS, for example, in the LOOKUP type 20 request to the CLIENTS file that appears in the TRANS file definition.

Commission_RateNUM 99.00

This field holds the commission rate charged to the client, and may be used during transaction in the calculation of the commission amount held in the TRANS file.

Photo EXT 30

The Photo.CLIENTS field is an external field, designed to store the name of an external image file. The "photos" files in the application are stored in .PCX format in the same directory as the CLIENTS file, and by convention take the Lastname.PCX as filename. The image file referred to in Photo.CLIENTS is automatically displayed on the second data page of the form STKC when accessed through the MENU program.

The COUNTRY File

The COUNTRY file contains details of countries in which CLIENTS may be located. Maintenance and browsing of the COUNTRY file is carried out from the File menu. The MENU program chains to STKFO. STKFO maintains both

COUNTRY and CURRENCY files. The COUNTRY file maintenance form is STKO.

The CLIENTS file uses COUNTRY as a LOOKUP file, ensuring that each CLIENTS record contains a valid country key. Name.COUNTRY is referred to in the LOOKUP validation attached to Country.CLIENTS.

Name	TXT IXU	30
Dialcode	NMI	999.
GMT Zone_Min	NMI VAL	+99.9
GMT Zone_Min.COUNTRY > - 13 AND GMT Zone_Max.COUNTRY < 13		
GMT Zone_Max	NMI VAL	+99.9
GMT Zone_Max.COUNTRY > - 13 AND GMT Zone_Max.COUNTRY < 13		

These two integer fields are validated to ensure that values entered are in the range -12 to +12.

The STOCKS File

The STOCKS file contains details of stocks in which the trading company trades. Stocks are identified primarily by Stock_Key, which is derived by concatenating Exchange and Symbol.

The STOCKS file maintenance function is accessed from the File menu, and can also be accessed from the CLIENTS or TRANS maintenance functions. The MENU program includes a section labeled **stkf** responsible for carrying out these functions. The STOCKS display form is STKS and the print form STKS2.

Pricing information includes the Price Date, Time, the Price Asked and Bid, and may be updated on a daily basis using the STKDC program accessed from the Daily menu in conjunction with the form STKDS.

Included in the STOCKS file is a Currency field. When creating or editing the Currency field, Superbase performs a LOOKUP to a separate CURRENCY file. This ensures that every STOCKS record contains a valid key for Currency. Prior to amendment or deletion of CURRENCY records, a check is programmed to ensure that no related STOCKS records exist (see STKFO). Combined with the use of LOOKUP during creation, this check ensures the integrity of the reference from STOCKS to CURRENCY files.

STOCKS File Definition

Symbol	TXT IXD	10
--------	---------	----

This field holds the symbol allocated by the exchange to the stock being traded. It is normally an abbreviation of the company name.

Exchange	TXT	16
----------	-----	----

Stock_Key VTX CLC RDO IXD 27
 Exchange.STOCKS + " " + Symbol.STOCKS

This virtual calculation creates a key for the stock record by concatenating the Exchange field, a space and the Symbol field. This key sequence ensures that all stocks quoted on the same exchange may be viewed and listed together. The Stock_Key is the link between the TRANS file and the CLIENTS file and is referred to by a LOOKUP in the TRANS file. Prior to deletion of a STOCKS record, a programmed check is carried out in the **stkf** routine to ensure that no related TRANS records exist.

Company Name	TXT	36
Sector	TXT IXD	30
Description	TXT	40
Stock_Type	TXT	16
Performance	TXT	40
Market	TXT	20

Some exchanges like London have a number of markets e.g. USM, Third.

Currency TXT VAL IXD 3
 LOOKUP (Currency.STOCKS,Currency.CURRENCY) ELSE REQUEST
 "Currency not found - Select another currency", "",20,, Currency.STOCKS,
 40, Currency.CURRENCY, Description.CURRENCY

The CURRENCY file contains currency details including the pricing factor and the exchange rate to the US Dollar. The equivalent field to Currency.STOCKS in the CURRENCY file is Currency.CURRENCY. On creation of a new STOCKS record, the LOOKUP is performed and, if the value entered for Currency.STOCKS is not found in Currency.CURRENCY, a dialog is displayed to force entry of a valid currency code.

Nominal Stock NUM ,9999999999.
 Current_Stock NUM VAL ,9999999999.
 Current_Stock.STOCKS <= Nominal Stock.STOCKS

This field holds the number of shares currently outstanding and is used in calculating the capitalization, see Cap Million below. The number of shares outstanding is validated as less than or equal to the number of shares authorized.

Price Date	DAT CON	mmm dd,yy
TODAY		
Price Time	TIM CON	hh:mm am
NOW		
Price Asked	NUM	9999.9999
Price Bid	NUM	9999.9999

During entry of new TRANS records, a default transaction price may be derived as either the Price Asked or the Price Bid, depending upon the Trans_Type: "b" for bought gets the Price Asked; "s" for sold gets the Price Bid.

Price Middle NUM CLC RDO 9999.9999
(Price Asked.STOCKS + Price Bid.STOCKS) / 2

Price Middle is a calculation that takes the average of the Price Asked and the Price Bid. This is used, for example, in comparing against previous Year_High and Year_Low prices in the new Year_High and new Year_Low reports and their associated updating routines.

Cap Million NUM CLC RDO ,99999.999
LOOKUP (Currency.STOCKS,Currency.CURRENCY) ?
Current_Stock.STOCKS * Price Middle.STOCKS / (1000000 *
Factor.CURRENCY): Cap Million.STOCKS

Cap Million.STOCKS is a self-referencing ternary calculation. It first performs a LOOKUP to the currency file to ensure that the Factor.CURRENCY used in the calculation is the Factor for this stock's currency. If the LOOKUP is successful, the Cap Million is calculated as the Current_Stock * Price Middle, using the appropriate scaling factor. Otherwise it is left unchanged.

Year_High NUM 9999.9999
Date of High DAT mmm dd,yy
Year_Low NUM 9999.9999
Date of Low DAT mmm dd,yy

These values are updated automatically by the update routines incorporated within MENU and accessed from the Update menu.

FYE_month NMI VAL 00.
FYE_month.STOCKS > 0 AND FYE_month.STOCKS < 13

The validation checks for a valid month number in the range 1 to 12.

Last_Net Profit NUM -,9999999999.
PE ratio NMI CLC RDO -999.
(Cap Million.STOCKS * 1000000) / Last_Net Profit.STOCKS

The Price Earnings ratio is obtained by dividing the capitalization (converted from millions back to units of currency) by the Last_Net Profit.

Address_1 TXT 30
Address_2 TXT 30
City TXT 30
State TXT 30
Zip Code TXT 10
Country TXT IXD 24

Telephone	TXT	20
Comments	TXT	40
FYE Stock	NUM	,9999999999.

Current_Stock.STOCKS

This field holds the number of shares outstanding at last fiscal year end, and is used in the calculation of Last_Earnings per share below.

Last_Earnings VNU CLC RDO -999.9999
 LOOKUP (Currency.STOCKS,Currency.CURRENCY) ? (Last_Net
 Profit.STOCKS * Factor.CURRENCY) / FYE Stock.STOCKS:
 Last_Earnings.STOCKS

This virtual calculation first performs a LOOKUP on the CURRENCY file, then uses the Factor to extend the Last_Net Profit before dividing by the number of shares outstanding at last fiscal year end. The Last_Earnings is accordingly expressed in pricing units e.g. pence.

The CURRENCY File

The CURRENCY file contains details of currencies in which stocks stored in the STOCKS file are quoted. Maintenance and browsing of the CURRENCY file is carried out from the File menu. The MENU program chains to STKFO. STKFO opens the form STKU.

An abbreviated form of the CURRENCY maintenance function is available from the Daily menu to provide a quick means of entering currency exchange rates. This is one of the functions performed by the STKDC program and uses the form STKDC.

Both the STOCKS file and the TRANS file use CURRENCY as a LOOKUP file, ensuring that each STOCKS record refers to a valid currency, and that factors used in TRANS calculations are correct. Prior to deleting a CURRENCY record a programmed check is carried out in STKFO to ensure that no related STOCKS records exist.

CURRENCY File Definition

Currency	TXT IXD	3
Description	TXT	30
Units	TXT	8

This refers to the pricing units, for example pence, in which stocks are quoted, rather than the currency units themselves, for example STG.

Factor	NMI	999.
--------	-----	------

Some currencies such as sterling have associated stock prices quoted in a unit other than the currency unit itself, for example pence. The Factor integer is the divisor for converting prices into the currency unit.

USD Xrate	NMI	9999.0000
Date of Xrate	DAT	mmm dd,yy

The latest exchange rates may be entered on a daily basis from STKDR. This routine automatically sets the Date of Xrate to TODAY. This date may be amended along with other CURRENCY fields by the maintenance routine STKFO.

The TRANS File

The TRANS file contains details of sale and purchase transactions that the trading company makes on behalf of its clients. The TRANS file maintenance function is accessed from the File menu, and may also be accessed directly from the CLIENTS or STOCKS maintenance forms. The MENU program includes a section labeled **stkf** responsible for carrying out these functions. The TRANS maintenance form is STKT.

TRANS file records may also be viewed by client or by stock on the third display page of the CLIENTS or STOCKS maintenance form, and by client on the Statements forms STKOS and STKOS2, accessible from the Documents menu.

When creating or editing a TRANS record, Superbase checks the Stock_Key by means of a LOOKUP to the STOCKS file, and the Customer_Ref by means of a LOOKUP to the CLIENTS file. Thus, the file definition for the TRANS file ensures that each transaction record refers to valid stock and client records. TRANS records also contain a number of fields which may be derived from values held in the associated STOCKS, CLIENTS and CURRENCY records. These include the Currency, Stock_Price, USD_Xrate and Commission_Rate.

TRANS File Definition

```

Stock_Key      TXT VAL REQ IXD  27
LOOKUP (Stock_Key.TRANS,Stock_Key.STOCKS) ELSE REQUEST
"Stock key not found - Please select another
Stock", "",20,,Stock_Key.TRANS, 60,Stock_Key.STOCKS,Company
Name.STOCKS

```

The Stock_Key is a required field. The LOOKUP validation and REQUEST dialog ensure that a valid Stock_Key is entered for each transaction.

```

Customer_Ref   TXT VAL REQ IXD  8
LOOKUP (Customer_Ref.TRANS,Customer_Ref.CLIENTS) ELSE
REQUEST "Customer ref not found - Please select another
customer", "",20,, Customer_Ref.TRANS, 60, Customer_Ref.CLIENTS,
Client.CLIENTS, Location.CLIENTS

```

The Customer_Ref is also a required field. The LOOKUP and REQUEST ensure that a valid Customer_Ref is entered for each transaction.

```

Transaction ref  NML CON RDO IXD  000000.

```

```

SER ("TRANS")
Trans_Type      TXT VAL REQ      1
Trans_Type.TRANS = "b" OR Trans_Type.TRANS = "s" ELSE REQUEST
"Valid types are (b)uy or (s)ell", "", 4,, Trans_Type.TRANS, 1
Trans_Date      DAT CON          mmm dd,yy
TODAY
Trans_Time      TIM              hh:mm am
Settlement_Date DAT CON          mmm dd,yy
Trans_Date.TRANS + 7

```

The Settlement_Date is a constant, set to (Trans_Date + 7). Since constants are evaluated once at the beginning of data entry for a new record, this definition is effectively equivalent to (TODAY + 7). However, as it is not read-only, it can be edited along with Trans_Date and Trans_Time.

```
Quantity      NML              99999999.
```

The quantity is the basis of the value calculations that follow.

```

Stock_Price    NUM CLC          99999999.0000
Stock_Price = 0 AND LOOKUP (Stock_Key.TRANS, Stock_Key.STOCKS)
AND (Trans_Type.TRANS = "s") ? Price Bid.STOCKS: Stock_Price.TRANS
= 0 AND LOOKUP (Stock_Key.TRANS, Stock_Key.STOCKS) AND
(Trans_Type.TRANS = "b") ? Price Asked.STOCKS: Stock_Price.TRANS

```

This actually says the following, "If the Stock_Price is zero AND if there exists a valid STOCKS record AND the transaction is a sale, then set the Stock_Price equal to the Price Bid from STOCKS; otherwise if the first two conditions are still true AND the transaction is a purchase then set the Price equal to the Price Asked from STOCKS; otherwise if neither of the sets of conditions applies then set the Stock_Price equal to itself."

The LOOKUPS are included here simply to ensure that the correct STOCKS record is being referred to under all circumstances. Part of the purpose of the definition is to allow the user to enter an override price during record creation. If the user enters a value, the calculation will leave it alone. If the user presses ENTER, the Stock_Price will be zero. The calculation will then set it to the Price Bid or Price Asked depending upon the value of Trans_Type.

```

Currency      TXT CLC RDO IXD  3
Currency.TRANS = "" ? Currency.STOCKS: Currency.TRANS

```

Currency.TRANS is another self-referencing ternary calculation. If the user presses ENTER, the value will be "", and will accordingly be set to Currency.STOCKS by the calculation. Otherwise it will be left alone.

```
Currency_Value NUM CLC RDO      99999999.00
```


$$\text{(Trans_Type = "b")} ? (\text{Stock_Price.TRANS} * \text{Quantity.TRANS}) /$$

$$\text{Factor.CURRENCY: } (\text{Stock_Price.TRANS} * \text{Quantity.TRANS}) /$$

$$\text{Factor.CURRENCY} * - 1$$

For a purchase, the Currency_Value is the Stock_Price times the Quantity divided by the Factor from the CURRENCY file. For a sale, its value is made negative. The reason for this is that the Currency_Value and USD Value (see below) are summed in reports to determine the balance owing to (+) or owed by (-) the trading company.

USD Xrate NUM CLC RDO 9999999.0000
 USD Xrate.TRANS = 0 AND LOOKUP
 (Currency.TRANS,Currency.CURRENCY) ? USD Xrate.CURRENCY:USD
 Xrate.TRANS

This calculation definition would allow for an override, but for the fact that it is set to read-only. The calculation performs a LOOKUP to ensure that the correct CURRENCY record is in memory, then sets the value of Xrate.TRANS to the exchange rate from the CURRENCY file. This field is used in calculating the USD Value (see below) of the transaction.

USD Price NUM CLC RDO 9999999.00
 (Stock_Price.TRANS / Factor.CURRENCY) / USD Xrate.TRANS

The USD Price calculation determines for memo purposes the dollar equivalent of the local currency Stock_Price.

USD Value NUM CLC RDO -999999.00
 Currency_Value.TRANS / USD Xrate.TRANS

The USD Value is the Currency_Value divided by the currency exchange rate.

Commission_Rate NUM CLC 99.00
 Commission_Rate = 0 AND LOOKUP
 (Customer_Ref.TRANS,Customer_Ref.CLIENTS) ?
 Commission_Rate.CLIENTS:Commission_Rate.TRANS

This definition allows for an override during record entry. If the user enters a value, it will be left alone. Otherwise if the user presses ENTER, the calculation performs a LOOKUP to ensure that the correct CLIENTS record is in memory, then sets the value of Commission_Rate.TRANS to the commission rate from the CLIENTS file.

Comm_Value NUM CLC RDO -9999999.00
 Commission_Rate.TRANS * ABS (USD Value.TRANS) / 100

The Comm_Value is calculated as the Commission_Rate multiplied by the USD Value of the transaction. The ABS function is used since the commission value is always owed to (+) the trading company.

IndexCustStock VTX CLC RDO IXD 35

Customer_Ref.TRANS + Stock_Key.TRANS
 IndexStockCust VTX CLC RDO IXD 35
 Stock_Key.TRANS + Customer_Ref.TRANS

These two fields are virtual read-only calculations that concatenate the Customer_Ref with the Stock_Key and vice versa. As indexed fields, they enable browsing through transactions by stock within client and vice versa.

Certificate_No TXT 20

This field holds the number of the stock certificate received in respect of a stock purchase made on behalf of a client. Certificate numbers may be entered daily using the STKDC routine, which opens the form STKDN. This form displays details of all TRANS purchase records where the Cert_Sent_yn flag is still "n" and enables input of Certificate numbers for these records.

Certificate advices may then be produced for all transactions where a certificate number has been newly entered, using the **m31** routine in the MENU program accessed from the Documents menu. This routine creates a temporary file CERTS, then produces certificate advices using the MERGE TEXT command on LECERTS.SBT.

Note

The activities of – entering a new transaction, editing it or adding a certificate number, producing certificate advices and finally updating the production flag – must be carried out in the correct sequence. Accordingly as each activity is completed, menu options are grayed or activated, and flags updated in CTRL to indicate the next valid activity or activities in the sequence to be carried out.

Cert_Sent_yn TXT CON 1
 "n"

The Cert_Sent_yn flag is a constant set to "n" in transaction entry. It is subsequently set to "y" following production of certificate advices for transactions where a certificate number has been newly entered. The flag update activity (**m43** of MENU) is accessed from the main Update menu, and must be carried out before further transactions or certificate numbers can be entered.

Invoice_Sent_yn TXT CON 1
 "n"

The Invoice_Sent_yn flag is a constant set to "n" in transaction entry. It is subsequently set to "y" following production of contract advices for all new transactions. The contract advice production (**m31** of MENU) and flag update (**m44** of MENU) activities are accessed from the main Documents and Update menus. The flag update activity must be carried out before further transactions can be entered.

Trans_Desc VTX CLC RDO 6
 (Trans_Type.TRANS = "b") ? "Bought": "Sold"

The above is a virtual field used in the preparation of the temporary file INVS by **m31** within MENU, and printed on the contract advices created using MERGE TEXT on LEINVS.SBT under program control.

Total_Due NUM CLC RDO z-9999999.00
USD Value.TRANS + Comm_Value.TRANS

This field is used in the preparation of the temporary file INVS and in the Trading Balance report programs STKRR and STKRB.

The CASH File

The CASH file contains details of cash receipts and payments entered through the Daily menu options Cash Receipts and Cash Payments. For both options the MENU program chains to STKDR. This routine opens and controls the form STKDR.

Batch_Ref NML 00000

In STKDR the last used Batch_Ref is taken from the file CTRL and incremented by 1.

Batch_Date DAT CON mmm dd,yy
TODAY

The Batch_Date is set by default to TODAY.

Client TXT VAL IXD 8
LOOKUP (Client.CASH, Customer_Ref.CLIENTS) ELSE REQUEST "Client
reference not found", "Select one of the following", 20,, Client.CASH,
60, Customer_Ref.CLIENTS, Client.CLIENTS, Location.CLIENTS

The LOOKUP and associated REQUEST ensure that each CASH record refers to a valid CLIENTS record.

Trans_Date DAT CON mmm dd,yy
Batch_Date.CASH

The Trans_Date is set by default to the Batch_Date.

Trans_Time TIM CON hh:mm am

The Trans_Time is set by default to NOW. In a fast-moving trading environment it is useful to be able to order transactions by time as well as date.

Trans_Type TXT VAL 1
Trans_Type.CASH = "r" OR Trans_Type.CASH = "p"

This is set automatically during the operation of STKDR.

Trans_ref NML 0000.
Trans_key VNL CLC RDO IXU 00000000.

Batch_ref.CASH * 10000 + Trans_ref.CASH

This virtual calculated field consists of the batch number followed by the transaction number and provides a unique reference for the transaction within the system. The file is indexed on this field and the Cash Audit Report is produced in this sequence.

Cash_ref	TXT	12
Amount	NUM	-999999999.00
Printed_yn	LOG	1

This flag is set to "n" when the record is created and subsequently updated to "y" by the Cash Audit Report following a successful print run.

Client_Net	VNU CLC RDO	-9999999.00
------------	-------------	-------------

LOOKUP (Client.CASH, Customer_Ref.CLIENTS) ?
 Net_Due.CLIENTS:Client_Net.CASH
 Trans_Desc VTX CLC RDO 8
 Trans_Type.CASH = "r" ? "Receipt": "Payment"
 Signed_Amt VNU CLC RDO -9999999.00
 Trans_Type.CASH = "p" ? Amount.CASH * -1: Amount.CASH

This virtual calculation is a ternary function that multiplies the transaction amount by -1 if the transaction is a purchase. Including this in the file definition ensures that the correct signed amount is always available for updating the CLIENTS Cash_Receipts amount during cash transaction creation, amendment or deletion.

The CONTROL File

The CTRL or control file contains flags to indicate whether particular routines can be run at the current point in chronological sequence. Routines that cannot be run are grayed on the main menu. The purpose is to enable a set of linked production and update processes, for example the production of certificate advices and the updating of flags in the TRANS file, to be spread over more than one working session.

CTRL File Definition

Field name	Attributes	Format	Routine	Initial Value
CERT	NMI IXD	9.	Documents Certificates	1
CONT	NMI	9.	Documents Contract Notes	1
CERU	NMI	9.	Update Certs Sent	0
CONU	NMI	9.	Update Notes Sent	0
TRAN	NMI	9.	File Transactions	1
CENT	NMI	9.	Daily Certificate Nos	1

The CTRL record is read at the beginning of execution of the MENU program. Each of the above flags has a value 0 or 1, corresponding to 0 for grayed mode, 1

for normal mode. As the relevant routines complete, the CTRL flags are updated and the menu items re-displayed with their new status.

For example, on completion of Certificate Advice Production, CERT, TRAN and CENT are set to 0 and CERU to 1. On completion of Certs Sent Update, CERU is set to 0 and CERT, TRAN and CENT to 1.

BATCH NMI 99999.

This field is incremented by the cash posting routine STKDR.

Activities

Menu Selection

The Trading system activities are all accessible from the master MENU program. This program displays a banner form and makes available a set of pull-down menus. For each menu item the program assigns a keyboard equivalent, allowing you to select an item either by releasing the right-hand mouse button when the required item is highlighted or by holding down the right-hand AMIGA key while you press the equivalent key. The program carries out the designated functions, then returns to await the next menu selection.

Once MENU is running, the system retains program control of the user's access to Superbase functions and facilities until the Exit option is selected from the File menu. The user can only perform the functions enabled by MENU and the other programs to which it chains. The system can therefore be considered as a 'sealed' application.

The set of programs (if necessary encrypted), forms, file definitions and other files making up the application can be run using only the Superbase Runtime version (or using Superbase 2). The application user does not then need to learn about defining files, forms or reports, but can focus on using the programs supplied. With the Runtime version, files can only be edited and updated under the control of the application programs and forms. The developers are thus able to retain responsibility for integrity of the application.

Some menu items accessible under MENU control give rise to a CHAIN to another program, while others access functions contained within MENU itself. All chained programs chain back to MENU on completion.

Grayed Menu Items

Certain menu items relevant to the production of transaction-related documents may be grayed. There is a cycle of activities associated with certificate advices and contract notes - transactions are entered and edited, documents produced, and transaction flags updated. The graying and re-activation of menu items is an effective way of controlling the sequence in which such separately defined activities are carried out. For example the File Transactions option is grayed after Certificate Advices have been produced. This prevents new records being created until updating has been carried out. As a further safeguard, the flags controlling the graying are held in a disk file CTRL, so that they remain the same from one session to the next. File Transactions is re-activated only following successful completion of the Update process.

The MENU Program

It is good programming practise to use a common style for all programs:

```
REM  MENU
REM  Superbase Demonstration Trading System
REM  Master Menu Program
REM  Last updated Feb 01, 91
```

The first statement sets the error trap:

```
ON ERROR GOTO ferror
```

The variable `ret%` will take an initial value of 0, hence first time through the routine executes the subroutine **sinit** (see below). If **MENU** is chained by another routine, `ret%` will have been set to 1. In this case, the routine displays the form in `banner$` and checks the previous menu selection in `ma%`. If it was from one of the first three columns, the menu is redisplayed by the **smenu** routine (see below). **smenu** is part of **sinit**.

```
mloop: REM Menu item selection
      MOUSE OFF
      IF ret% THEN
        REM Returning from another routine
        IF FORM <> banner$ THEN OPEN FORM banner$ ELSE FORM
        IF ma% < 4 THEN GOSUB smenu
      ELSE
        REM First time thru
        GOSUB sinit
      END IF
```

Menu Item Selection

The following is the master loop in which the program waits for a menu selection. The menu, having been assigned in **smenu**, is now switched on and the `ss1$` status message, assigned in **sinit**, is displayed. A **WHILE WEND** loop is used so that the routine can check whether the minute has changed. If it has, the **shead** routine is called to refresh the header bar display with the current date and time (see below). If this was not required, then a **WAIT MENU** could be used in place of the **WHILE WEND** loop.

```
MENU ON ma%,mb%
SET STATUS ss1$
WHILE NOT ma%
  IF h2% <> MINS ( NOW ) THEN GOSUB shead
WEND
IF ma% < 4 THEN MENU CLEAR
ret% = 1
ON ma% GOTO m1,m2,m3,m4,m5,m6
```

The six possible `ma%` values correspond to the six menu columns. Similarly the `mb%` values correspond to the individual menu options within the column.

Initialization

sinit carries out the functions associated with entering the system for the first time. It closes all files and forms, turns the browsing panel off and turns the status bar on. It displays a status message and performs a `SET REQUEST HEADING` to provide a custom heading for all `REQUEST` dialogs and error messages.

```
sinit:  REM Initialize first time thru
        CLOSE ALL :PANEL OFF : SET PAGING ON :STATUS ON :NUMBASE
        "z-99999.00"
        h3$ = "Superbase Demonstration Trading System "
        SET STATUS "Initializing application"
        SET REQUEST HEADING "Trading System"
```

The routine calls **shead** to display the header bar, opens the banner form, then opens the `TRANS` and `CTRL` files. The `TRANS` file includes `LOOKUPs` in its file definition to the `CLIENTS`, `STOCKS` and `CURRENCY` files, hence these files will automatically be opened at the same time. As `CLIENTS` includes a `LOOKUP` to `COUNTRY`, this file will also be opened. Opening a file makes the first record in the first index sequence the current record. The `CTRL` file contains a single record.

```
GOSUB shead
OPEN FORM banner$
OPEN FILE "TRANS":OPEN FILE "CTRL":REM opens all files
```

The next section dimensions system arrays and assigns `REQUEST` dialog messages and `STATUS` messages to string variables:

```
DIM pstk$(12),pstk%(12):REM key arrays for multi-record screens

REM Dialog messages
ms1$ = "Is entry correct?"
ms2$ = "Continue with data entry?"
ms3$ = "Do you wish to reprint?"
ms4$ = "Has all output been correctly printed?"
etc.
REM Status line messages
ss1$ = "Select a pulldown menu option"
ss2$ = "Select a pushbutton option"
ss3$ = "Enter data for new record"
ss4$ = "Use TAB or click on field to be edited. ESC to stop."
etc.
```

The next section clears any previous menu assignment. The **smenu** section defines the menu. Notice the use of "&" to indicate `AMIGA+`... keyboard equivalents and the use of the `CTRL` flags to give grayed or active status to certain menu items:

MENU CLEAR

smenu: REM Define pulldown menus

MENU 1,0,1,"&File"

MENU 1,1,1,"&Clients"

MENU 1,2,1,"&Stocks"

MENU 1,3,1,"C&ountries"

MENU 1,4,1,"C&urrencies"

MENU 1,5,tran.CTRL,"&Transactions"

MENU 1,6,1,"&Exit"

MENU 2,0,1,"&Daily"

MENU 2,1,1,"&Currency Rates"

MENU 2,2,1,"&Stock Prices"

MENU 2,3,cent.CTRL,"Certificate &Nos"

MENU 2,4,1,"Cash &Receipts"

MENU 2,5,1,"Cash &Payments"

MENU 3,0,1,"D&ocuments"

MENU 3,1,cert.CTRL,"&Certificates":

MENU 3,2,cont.CTRL,"C&ontract Notes"

MENU 3,3,1,"&Statements"

MENU 4,0,1,"&Update"

MENU 4,1,1,"Stock &Highs"

MENU 4,2,1,"Stock &Lows"

MENU 4,3,ceru.CTRL,"&Certs Sent"

MENU 4,4,conu.CTRL,"&Notes Sent"

MENU 4,5,1,"&Reset"

MENU 5,0,1,"&Query"

MENU 5,1,1,"&Run"

MENU 5,2,1,"&New"

MENU 5,3,1,"&Open"

MENU 5,4,1,"&Edit"

MENU 5,5,1,"&Save"

MENU 6,0,1,"&Reports"

MENU 6,1,1,"&Traded Stocks"

MENU 6,2,1,"Stock &Highs"

MENU 6,3,1,"Stock &Lows"

MENU 6,4,1,"T&ading Balances"

MENU 6,5,1,"-----"

MENU 6,6,1,"Telephone &Nos"

MENU 6,7,1,"Client &Balances"

MENU 6,8,1,"&Cash Audit"

RETURN

Header, Date and Time Display

The **thead** subroutine is called once from **sinit** and again at one minute intervals during the WHILE...WEND loop used for menu item selection. This routine formats a line of text consisting of the system date, the application title and the system time, and displays it as the heading.

```
thead: REM Set heading every minute during menu wait
      h1$ = DATE$ ( TODAY ,"mmm dd, yy"):h2$ = TIME$ ( NOW ,"hh:mm
      am")
      h2% = MINS ( NOW ):h3% = (46 - LEN (h3$)) / 2
      h4$ = h1$ + SPACE$ (h3%) + h3$ + SPACE$ (h3%) + h2$
      SET HEADING h4$
      RETURN
```

Modular Structure

The menu selection routine yields values for **ma%** and **mb%** corresponding to the menu column and item within column. The MENU program follows a modular structure with routines labeled to correspond to their **ma%**, **mb%** values:

m1 indicates the first menu column, **m11** the first item within the column, **m111** the first labeled command within the **m11** activity section, and so on.

File menu

The first menu column consists of the File menu options. The Clients, Stocks and Transactions options are handled by the MENU program **stkf** routine. The Country and Currency options are handled by a separate program **STKFO**, and the remaining Exit option is handled at **m16**:

```
m1:   REM ma%= 1 File menu options
      selkey$ = "":selkey% = 0
      ON mb% GOTO stkf,stkf,m13,m13,stkf,m16
m13:  REM File Countries, Currencies
      CHAIN "stkfo"
m16:  REM File Exit
      CLOSE ALL :SET HEADING "":SET REQUEST HEADING "":PANEL
      ON :END
```

Daily Menu

There are five activities listed in the Daily menu. The first three, namely Currency Rates, Stock Prices and Certificate Nos, are controlled by the program **STKDC**, while the remaining two, namely Cash Receipts and Cash Payments, are carried out by the program **STKDR**. Please see below under the relevant activity sections for a description of these programs.

```
m2:    REM ma%=2 Daily menu options
      ON mb% GOTO m21,m21,m21,m24,m24
m21:   REM Daily Currency rates, Stock prices, Certificate numbers
      CHAIN "stkdc"
m24:   REM Daily Cash receipts, Cash payments
      CHAIN "stkdr"
```

Document Menu

As all the Documents options are executed from within MENU, it is possible to use a GOSUB rather than a GOTO structure:

```
m3:    REM ma%=3 Documents menu options
      ON mb% GOSUB m31,m31,stkf
      GOTO mloop
```

Please see below under the relevant activity sections for an annotation of the **m31** and **stkf** parts of MENU.

Update Menu

Following the **m31** routine in MENU, you will find a similar GOSUB structure for the Update options:

```
m4:    REM ma%=4 Update menu options
      ON mb% GOSUB m41,m42,m43,m44,m45
      GOTO mloop
```

Query Menu

The **m5** routine is a useful example of how to provide users with an ad hoc Query facility under program control. **m5** controls access to the query functions Run, New, Open, Edit and Save, via a CASE structure with **mb%** as the case variable. There are a number of stored .SBQ Query files shipped with the application. These were created with NUMBASE set to "Z-9999999.00", hence:

```
m5:    REM ma%=5 Query menu options
      MOUSE ON :NUMBASE "z-9999999.00"
      SELECT CASE mb%
```

Query Run calls the subroutine **sdorp** to request selection of the display or printer device, then uses ? QUERY to output the existing query to the selected device. Following output, the FORM command re-displays the banner form.

```
CASE 1:REM Query Run
      GOSUB sdorp
      IF a% THEN
        IF p$ = "p" THEN
          ? QUERY TO PRINTER
        ELSE
```

```

? QUERY
SET STATUS ss14$
WAIT KEY OR MOUSE
END IF
FORM
END IF

```

Query New sets up a specific error trap at **mqerror**, clears any existing query and activates the Query Edit dialog. Following the CASE loop, the routine re-sets mb% from 2 to 1 and re-enters the CASE loop, so that Query New is automatically followed by Query Run.

```

CASE 2:REM Query New
SET STATUS ss15$
ON ERROR GOTO mqerror
NEW QUERY
EDIT QUERY

```

Query Open displays a type 14 dialog to request selection of a valid .SBQ filename, then loads the selected query.

```

CASE 3:REM Query Open
REQUEST ms23$, "", 14,a%,q$
IF a% THEN LOAD QUERY q$

```

Query Edit sets up the error trap at **mqerror**, displays a status message, then activates the Query Edit dialog.

```

CASE 4:REM Query Edit
SET STATUS ss15$
ON ERROR GOTO mqerror
EDIT QUERY

```

Query Save displays a type 14 dialog to request input of a query filename, then saves the current query to that filename with its .SBQ extension.

```

CASE 5:REM Query Save
REQUEST ms24$, "", 14,a%,q$
IF a% THEN SAVE QUERY q$
END SELECT
IF mb% = 2 THEN mb% = 1:GOTO m5:REM if New query then Run

```

The query error trap at **mqerror** re-sets the general error trap and the NUMBASE before returning to the menu item selection loop.

```

mqerror: REM Trap for errors generated from Query dialog
ON ERROR GOTO ferror
NUMBASE "z-99999.00"
GOTO mloop

```

Report Menu

The final menu column enables access to the reports. Each of these is a separate program created using the Report Generator, then edited in various ways. An example of editing is the insertion of a CHAIN command to chain back to MENU following successful execution.

```
m6:    REM ma%=6 Reports menu options
      GOSUB sdorp: IF a% = 0 THEN GOTO mloop
      ON mb% GOTO m61,m62,m63,m64,mloop,m66,m67,m68
m61:  CHAIN "stkrte"
m62:  CHAIN "stkrhe"
m63:  CHAIN "stkrle"
m64:  CHAIN "stkrre"
m66:  CHAIN "stkrne"
m67:  CHAIN "stkrbe"
m68:  CHAIN "stkrce"
```

Selection of output device

The **sdorp** subroutine is accessed from the query section **m5** and from the reports section **m6** in order to provide the user with the ability to select the display or the printer as output device:

```
sdorp: REM Select d=display p=print
      p$ = "d"
      REQUEST ms17$, "", 4, a%, p$, 1
      IF NOT (p$ LIKE "[pd]") GOTO sdorp
      IF a% THEN
        p$ = LCASE$ (p$)
        IF p$ = "d" THEN SET STATUS ss7$
        ELSE SET STATUS ss6$:PRINT ;
      END IF
      RETURN
```

Error trapping

At the foot of the MENU program is the general error trap routine **fferror**. For error numbers other than 11 or 57, the routine displays the corresponding error message given by ERR\$ (ERRNO) and provides the option to RESUME at the menu selection loop. Error number 11 will occur if there is a break in the program, for example if the user presses CTRL C. In this case the option is given to RESUME at the next instruction. Error number 57 occurs during record creation or key amendment if the action would give rise to a duplicate key in one of the indexes. In this case the option is given to RESUME at the form pushbutton wait point. In all

cases the user can select Cancel in the error dialog, in which case MENU will open the program editor window and stop.

```
ferror:  REM General error trap
         res% = 0
         IF ERRNO = 11 THEN
             REQUEST ms21$,ms22$,130,a%:IF a% = 1 THEN res% = 1
         ELSE IF ERRNO = 57 THEN
             REQUEST ms8$,"",2,a%:res% = 2
         ELSE
             REQUEST ERR$ ( ERRNO ),ms9$,114,a%:
             IF a% = 1 THEN res% = 3
         END IF
         IF res% = 1 THEN RESUME
         IF res% = 2 THEN RESUME floop
         IF res% = 3 THEN RESUME mloop
         EDIT
```

File Maintenance

Clients

The CLIENTS maintenance activity is the first option on the Trading System File menu. It is controlled using the combination of the MENU routine **stkf** and the form STKC.

Form STKC

The STKC form comprises 4 pages and is designed to be used in conjunction with the **stkf** routine. The first page contains a single object, the invisible embedded command `SELECT FORM KEY selkey$`. The form is opened by **stkf**, using an `OPEN FORM` command. This command always sets the current record to the first record in the current index sequence. The purpose of the invisible command on the first form page is to re-set the current record using the value in `selkey$`, before any data becomes visible. **stkf** then issues a command `FORM 2` to display the second actual form page. This is the first display page and shows maintainable fields for the required record in the CLIENTS file. It also contains the Dialcode from the COUNTRY file. A link set up between `Country.CLIENTS` and `Name.COUNTRY` enables the correct Dialcode to be displayed.

This first display page includes nine command pushbuttons. The Exit button is situated top-right and the remaining buttons arranged in a control panel across the foot of the page. The commands attached to the buttons are as follows:

Exit	GOTO fexit
Enter	GOTO fenter
Edit	GOTO fedit
Delete	GOTO fdelete
P-2	FORM SHOW 3
P-3	FORM 4
Index	GOSUB sindex
Browse	GOSUB spanel
Print	GOSUB sprint

Pushbuttons are an effective and visually attractive alternative to pull-down menus or function keys for controlling access to functions associated with a form. Some pushbuttons access routines within **stkf**, while others, for example `FORM SHOW 3`, are self-contained.

The second display page of the form includes the virtual field Client and the external fields Photo and Notes. The `FORM SHOW 3` command attached to the "P-2" button ensures that the external objects themselves are displayed as rather than simply their file names. This form page includes the same pushbuttons as the first, with the exception of the following in place of "P-2":

P-1	FORM 2
-----	--------

This page also includes an invisible command, situated at the foot of the page and labeled stop\$: IF ed% THEN GOTO fedit2 ELSE GOTO fenter2. The function of this command is to terminate the data entry or edit process at the foot of this form page.

The third display page of STKC shows the Customer_ref, and Client fields from CLIENTS. It then shows a "one-to-many" transaction block of 10 lines, showing details from the TRANS file, including the Transaction ref, Stock_Key, Quantity, US\$ Value and Commission for the transaction. Behind each transaction line and stretching the width of the line is an invisible pushbutton. These are labeled ln1\$ through ln10\$. The commands associated with them are mp%=1 through mp%=10. Their use is described below. At the foot of the block are two form calculations summing the US\$ value and commission columns.

This page also includes an Exit pushbutton and a control panel of other pushbuttons arranged across the foot of the page. The commands attached to these buttons are as follows:

Trans	mb%=tra%:GOTO fswitch
Next	SELECT FORM NEXT PAGE:VIEW
Prev	SELECT FORM PREVIOUS PAGE:VIEW
P-1	FORM 2
P-2	FORM SHOW 3
Index	GOSUB sindex
Browse	GOSUB spanel
Stock	mb%=stk%:GOTO fswitch

The Trans and Stock pushbuttons are used in conjunction with the invisible pushbuttons behind each transaction line to switch to the STOCKS or TRANS maintenance activities, which are also controlled by **stkf**. The Trans and Stock pushbuttons set the menu column variable mb% to the corresponding value, then branch to the **fswitch** routine. This routine sets the line number variable mp% to 0, waits for another mouse click, and tests mp%. Clicking on one of the invisible line pushbuttons sets mp% to a line number in the range 1 to 10. Once mp% has a value in this range, the routine branches back to the start of **stkf** to display transaction or stock details for the transaction line selected.

Stocks

The STOCKS maintenance activity is the second option on the File menu. It is controlled using the combination of the MENU routine **stkc** and the form STKS.

Form STKS

The STKS form is a 4 page form, designed to be used in conjunction with **stkf**. Its format is similar to the CLIENTS form STKC. The first page contains the invisible embedded command `SELECT FORM KEY selkey$`. This re-sets the current record using the value in `selkey$` before any data becomes visible. The second form page is the first actual display page and shows maintainable fields for the required record in the STOCKS file. It contains the Dialcode from the COUNTRY file next to the company telephone number. A link set up between `Country.STOCKS` and `Name.COUNTRY` enables the correct Dialcode to be displayed.

The second display page of STKS shows further details from the STOCKS record, including the pricing information. It includes the currency description and units from the CURRENCY file. These are retrieved by means of a link set up between `Currency.STOCKS` and `Currency.CURRENCY`.

Field data in STKS is displayed in the STOCKS color of light blue. The first and second display pages of STKS include identical pushbuttons to those on STKC. The second display page also includes an invisible pushbutton to prevent entry or editing from continuing on to the third display page.

The third display page of STKS is again similar to the third display page of STKC. It shows the `Stock_Key` from STOCKS, together with a one-to-many block of 10 records from the TRANS file including the Date, Transaction ref, Client Name, Customer_Ref, Quantity, US\$ Value and Commission for each transaction. The client name is retrieved from the CLIENTS record by means of a link set up between `Customer_Ref.TRANS` and `Customer_Ref.CLIENTS`. Behind each transaction line is an invisible pushbutton, and at the foot of the block are two form calculations summing the US\$ value and commission columns.

The pushbuttons are identical to those on the equivalent page of STKC, with the exception of the following:

Client `mb%=cli%:GOTO fswitch`

The Trans and Client pushbuttons are used in conjunction with the invisible pushbuttons behind each transaction line to switch to the TRANS or CLIENTS maintenance activities.

Transactions

The Transactions or TRANS file maintenance activity is the fifth option on the File menu. It is controlled using the combination of **stkf** and the form STKT.

The Form STKT

STKT is a two page form, designed to be used in conjunction with **stkf**. The first page contains the invisible embedded command `SELECT FORM KEY selkey$`. This re-sets the current record using the value in `selkey$` before any data becomes visible. The second form page is the first actual display page and shows maintainable and calculated fields for the required record in the TRANS file.

Entering data for a new transaction provides a striking example of the power available at the file definition level. The Transaction ref, Trans_Date, Trans_Time and Settlement_Date are defined as constants. Values for these fields are accordingly displayed at the beginning of the data entry process. The field Trans_Type.TRANS has two possible values "b" or "s", each with an associated radio button and descriptive text object on the form. During data entry, the **fenter** routine sets Trans_Type by default to "b". The Stock_Key and Customer_Ref fields both have associated validation formulas that force entry of valid codes through type 20 request dialogs. Entry of a valid Stock_Key results in the display of the Company Name from the STOCKS file, retrieved through the link set up between Stock_Key.TRANS and Stock_Key.STOCKS. Entry of a valid Customer_Ref results in the display of the Client name from the CLIENTS file, retrieved through the link set up between Customer_Ref.TRANS and Customer_Ref.CLIENTS.

Following input of a Quantity, the Currency code is displayed from the related STOCKS record. The next field on the form is the Stock_Price. The calculation formula for this field enables either a value to be entered or the Price Bid or Price Asked to be retrieved from the STOCKS record, according to whether the transaction is a sale or purchase. The Stock_Price and Quantity are used to calculate the Currency_Value. The USD Xrate is extracted from the CURRENCY file and used to calculate the USD Price and USD Value. The next field, Commission_Rate, is defined so as to enable either a value to be entered or the Commission_Rate to be retrieved from the CLIENTS file. The Commission_Rate and USD_Value are used to calculate the Comm_Value, and the USD Value and Comm_Value are used to derive the Total_Due.

The form includes two check boxes with the possible values "y" or "n" for the fields Cert_Sent_yn.TRANS and Invoice_Sent_yn.TRANS. The form is color-coded. Data originating from the CLIENTS file appears in yellow, data from the STOCKS file in light blue, data from the CURRENCY file in dark blue, while values entered or calculated for the TRANS file appear in green.

STKT includes an Exit pushbutton and a control panel of other pushbuttons arranged across the foot of the form. The commands attached to these buttons are as follows:

Exit	GOTO fexit
Enter	GOTO fenter
Edit	GOTO fedit
Delete	GOTO fdelete
Index	GOSUB sindex
Browse	GOSUB spanel
Client	mb%=cli%:GOTO fswitch
Stock	mb%=stk%:GOTO fswitch

The Client and Stock pushbuttons are used to switch to the CLIENTS or STOCKS or maintenance activities, for display of full details of the related client or stock.

The stkf Routine

This is the generalized routine within MENU controlling the Clients, Stocks and Transactions options within the File menu and the Documents option within the Documents menu.

```
stkf:  REM File Clients, Stocks, Transactions
      REM Documents Statements
      fst% = 0:cli% = 1:stk% = 2:sta% = 3:tra% = 5:k% = 1:t% = 0
      REM Set option parameters
      SELECT CASE mb%
      CASE cli%:REM File Clients
        f$ = "clients":fkey$ = "Customer_Ref":fmd$ = "stkc":fmp$ = "stkc2"
```

A slightly different version of the STKC form, called STKCE, is used in conjunction with an EGA display:

```
      IF vga$ = "n" THEN fmd$ = "stkce"
      CASE stk%:REM File Stocks
        f$ = "stocks":fkey$ = "Stock_Key":fmd$ = "stks":fmp$ = "stks2"
      CASE sta%:REM File Transactions
        f$ = "clients":fkey$ = "Customer_Ref":fmd$ = "stkos":fmp$ = "stkos2"
      CASE tra%:REM Documents Statements
        f$ = "trans":fkey$ = "Transaction ref":fmd$ = "stkt":k% = 2
      END SELECT

      FILE f$
      INDEX fkey$
      IF FORM <> fmd$ THEN OPEN FORM fmd$
```

The keyword INDEX is used to retrieve the current indexed field. The following command stores it in the variable ind\$.

```
ind$ = INDEX
GOSUB sset
```

The **sset** and **ssearch** routines, found towards the end of the MENU program, use the \$\$ field indirection operator. fkey\$\$ is the actual key value currently assigned to the indexed field variable held in fkey\$. Since fkey\$ refers in all cases to a field with a unique index, the value held in ckey\$ or ckey% will always be unique for the file, and is later used in **ssearch** to make this record the current record again.

```
sset:  REM Stores unique value for record in key
      IF k% = 1 THEN ckey$ = fkey$$
      IF k% = 2 THEN ckey% = fkey$$
      RETURN
```

ssearch starts by finding the first record with a key value equal to that assigned to the indexed field variable held in ind\$. It then continues selecting records until it finds the record uniquely identified by ckey\$ or ckey%:

```
ssearch: REM Gets unique record even if index has duplicate keys
      SELECT FORM KEY ind$$
ssearch2: REM Loop until current record found
      IF k% = 1 THEN IF fkey$$ = ckey$ THEN RETURN
      IF k% = 2 THEN IF fkey$$ = ckey% THEN RETURN
      SELECT FORM NEXT
      GOTO ssearch2
```

To continue with the discussion of **stkf**, after the form has been opened, and unless it is a statement form, the first display page is actually page 2 of the form:

```
      REM File forms have command only on page 1 to select form on selkey
      IF mb% <> sta% THEN FORM 2
      GOTO fwait
floop:  REM Re-display form
      GOSUB sset
      GOSUB ssearch
      VIEW
fwait:  REM Wait here for form pushbutton
      ed% = 1
      SET STATUS ss2$
      WAIT MOUSE
      GOTO fwait
```

fwait is the point at which the program awaits a form pushbutton selection. for The command WAIT MOUSE awaits a mouse click. Clicking on a command pushbutton causes execution of the form command string associated with it. If this is a self-contained command, for example FORM 4, the program will then execute the next statement, namely GOTO fwait. Otherwise, if the form command string is a GOTO or GOSUB then command execution will continue at the specified label.

floop is the point to which the program returns after an Enter, Edit or Delete operation. It calls **ssset** and **sssearch** to ensure that the correct record is made the current record.

Entering a New Record

The Enter pushbutton accesses the **fenter** routine. This displays a blank form for input, assigns default values for the file being maintained and displays a status message. The ENTER 3,0 command enables fields to be input from field 3 in the form data entry sequence until there are no more editable fields on the form, or until the user presses ESC, or until the embedded command labeled stop\$ is encountered (IF ed% THEN GOTO fedit2 ELSE GOTO fenter2). If the entry is confirmed as correct, then new record is stored.

```
fenter: REM Enter pushbutton
      BLANK FORM: ed% = 0
      IF mb% = tra% THEN
        Trans_Type = "b":del% = 0:SET STATUS ss18$
      ELSE
        IF mb% = cli% THEN Class.CLIENTS = "P"
        SET STATUS ss3$
      END IF
      MOUSE ON :ENTER 3,0
fenter2:
      MOUSE OFF
      b% = 0
      REQUEST ms1$,"",127,a%
      IF a% THEN
        STORE :REM Store new record
```

For transaction records, the routine sets amt% equal to Total_Due and calls the **supdate** subroutine. The option is then given to enter details of the next record or to return to pushbutton control:

```
      del% = 0
      IF mb% = tra% THEN amt% = Total_Due.TRANS: GOSUB supdate
      REQUEST ms2$,"",130,b%
      ELSE
        SELECT FORM CURRENT :REM Re-display current form
      END IF
      IF b% THEN GOTO fenter
      FORM 2
      GOTO floop
```

supdate changes the current file to CLIENTS, then updates the Trading_Balance by amt% for the record whose key is in Customer_Ref:

```
supdate:REM When transaction changes, update balance in Client record
      SET STATUS ss17$
```

```

FILE "clients"
INDEX Customer_Ref.CLIENTS
SELECT KEY Customer_Ref
IF FOUND ("" ) THEN
    Trading_Balance.CLIENTS = Trading_Balance.CLIENTS + amt%
    STORE
ELSE
    REQUEST ms18$,"",100,a%
END IF
FILE f$
RETURN

```

Editing an Existing Record

The Edit pushbutton accesses the **fedit** routine. Prior to editing transaction records, the Total_Due is stored in oamt%. The ENTER 3,0 command works in a similar way to its equivalent in fenter. On termination of editing, if the entry is confirmed as correct, the edited record is stored:

```

fedit:  REM Edit pushbutton
        SET STATUS ss4$
        IF mb% = tra% THEN oamt% = Total_Due.TRANS
        MOUSE ON :ENTER 3,0
fedit2:
        MOUSE OFF
        REQUEST ms1$,"",127,a%
        IF a% = 1 THEN
            STORE: REM Stores edited data

```

For transaction records, the routine sets amt% equal to the difference between the new and old Total_Due and calls supdate. The routine then returns to **floop** to await the next pushbutton:

```

        IF mb% = tra% THEN amt% = Total_Due.TRANS - oamt%: GOSUB
supdate
        ELSE
            SELECT FORM CURRENT :REM Re-dispay form as before edit
        END IF
        GOTO floop

```

Deleting an Existing Record

The Delete pushbutton accesses the **fdelete** routine. Prior to deleting CLIENTS or STOCKS records, the routine tests whether any related TRANS records exist. If so, the routine sets the del% flag to 0, displays an appropriate message and returns to **fwait**. Otherwise, the routine asks for confirmation of deletion. Prior to deleting

TRANS records, The routine sets amt% to the negative of Total_Due and calls **supdate**.

```
fdelete: REM Delete pushbutton
del% = 1
IF mb% <> tra% THEN
    REM Clients, Stocks, test for related transactions, if so set del%=0
    SET STATUS ss16$
    FILE "trans"
    skey$ = ckey$
    IF mb% = cli% THEN INDEX Customer_Ref.TRANS
    IF mb% = stk% THEN INDEX Stock_Key.TRANS
    SELECT KEY skey$
    IF FOUND ("" ) THEN del% = 0
    ckey$ = skey$
    FILE f$
END IF
IF del% = 1 THEN
    REQUEST ms14$,"",119,a%
    IF a% THEN
        IF mb% = tra% THEN amt% = - Total_Due.TRANS:
        GOSUB supdate
        SELECT REMOVE
    END IF
ELSE
    REQUEST ms19$,ms20$,100,a%
END IF
IF del% = 0 GOTO fwait
SELECT FORM KEY ind$$
GOTO floop
```

Changing Index

The Index pushbutton accesses the **sindex** subroutine. This displays a REQUEST type 7 dialog showing the indexes for the file being maintained. Following selection of an index, the routine makes it the current index, then access the **ssearch** subroutine to re- locate the current record in the new index sequence.

```
sindex: REM Index pushbutton
REQUEST ms7$,"",7,a%,ind$
IF a% THEN
    INDEX ind$
    GOSUB ssearch
END IF
```

VIEW
RETURN

Browsing

The Browse pushbutton accesses the **spanel** subroutine. This sets up a special ON ERROR destination, and composes a status line message to inform the user of the sequence in which browsing is taking place. The routine uses WAIT PANEL to activate the browsing panel. The panel remains active until it is de-activated by means of the panel stop button or CTRL+C. At this point the routine re-sets the error trap, executes a SELECT WHERE to turn off any filter, calls **sset** to get the uniquely identifying key for the record, and returns for the next pushbutton selection:

```
spanel: REM Browse pushbutton
        ON ERROR GOTO serror
        err% = 0
        ss5$ = ss10$ + ind$ + ss11$
        SET STATUS ss5$
spanel2:
        WAIT PANEL
        ON ERROR GOTO ferror
        SELECT WHERE
        IF err% = 1 THEN BREAK ON :PANEL OFF
        GOSUB sset
        RETURN
serror: BREAK OFF :REM Error trap for Wait Panel errors
        REQUEST ERR$ ( ERRNO ), "", 2, a%
        err% = 1
        SELECT FORM CURRENT :RESUME spanel2
```

Switching between Clients, Stocks and Transactions

The Trans and Stock pushbuttons on display page 3 of STKC are used to switch to displaying full transaction or stock details for one of the client transactions, bypassing the main pulldown menu. The Trans and Client pushbuttons on display page 3 of the STKS form, and the Client and Stock pushbuttons on display page 1 of the STKT are used in a similar way. The Trans and Stock pushbuttons on STKC first set mb% to tra% or stk% then branch to **fswitch**. This sets mp% = 0, waits for a further mouse click and tests mp% for a value in the range 1 to 10. The invisible pushbuttons under the transaction lines on STKC have associated commands of the form mp% = 1, etc. The routine selects the corresponding row, sets selkey\$ to the corresponding key field value, then branches to **stkf**:

```
fswitch: REM Assign key from transaction and form/record
        mp% = 0
        SET STATUS ss21$
        WAIT MOUSE
```



```
IF (mp% = 0 OR mp% > 10) THEN GOTO fswitch
SELECT FORM ROW mp%
fswitch2:REM Entry from Trans form: no need to select line
SELECT CASE mb%
CASE cli%
    selkey$ = Customer_Ref.TRANS
CASE stk%
    selkey$ = Stock_Key.TRANS
CASE tra%
    selkey% = Transaction ref.TRANS
END SELECT
GOTO stkf
```

Countries

The COUNTRY maintenance activity is the third option on the File menu. It is controlled using the combination of the program STKFO and the form STKO.

Form STKO

In STKO, details for a number of COUNTRY records are arranged on a single form page, using the 'transaction' or duplicate line function of the Form Designer. STKO displays up to 10 lines of COUNTRY records, one per line. The form includes an invisible pushbutton positioned behind each line. These pushbuttons assign a line number in the range 1 to 10 to the variable mp%.

The form includes an Exit pushbutton and five function pushbuttons arranged in a control panel across the foot of the form. The commands attached to the buttons are as follows:

Exit	GOTO fexit
Enter	GOTO fenter
Delete	GOTO fdelete
Next Page	GOTO fnext
Prev Page	GOTO fprev
Print	GOTO fprint

There is no pushbutton for the Edit function. The user simply clicks on the line to be edited. The Next Page and Prev Page pushbuttons enable browsing forwards or backwards through the file 10 records at a time.

Currencies

The CURRENCY maintenance activity is the fourth option on the File menu. It is controlled using the combination of the program STKFO and the form STKU.

Form STKU

Details for a number of CURRENCY records are arranged on a single form, in a manner similar to the STKO form for COUNTRY maintenance. STKU displays up to 10 lines of CURRENCY records, one per line. Fields are framed by the CURRENCY color of dark blue. STKU pushbuttons are similar to those for STKO.

The Program STKFO

The program STKFO controls the maintenance function for both COUNTRY and CURRENCY files. STKFO is designed to be CHAINED from MENU. It accordingly assumes that arrays are dimensioned and that dialog and status messages have been assigned by the **sinit** routine in MENU. **ffld%** holds the number of maintainable fields for the file and **frow%** the number of lines on the form.

```

      REM  STKFO
      REM  Maintain Countries/Currencies
      REM  Last updated Feb 01, 91
      ON ERROR GOTO ferror
stkfo:  REM Assign function parameters
      cou% = 3:cur% = 4
      SELECT CASE mb%
      CASE cou%
        f$ = "country":fkey$ = "Name.COUNTRY":fmd$ = "stko":fmp$ = "stko2"
        ffld% = 4
      CASE cur%
        f$ = "currency":fkey$ = "Currency.CURRENCY":fmd$ = "stku":fmp$ =
"stku2"
        ffld% = 6
      END SELECT
      frow% = 10:fst% = 0:OPEN FORM fmd$

```

The program returns to **floop** after entering one or more new records. The **slimits** subroutine re-establishes the first and last keys in **fstk\$** and **lstk\$**. It branches to **fform** to display from the first record on file:

```

floop:  REM Display from first record
      GOSUB slimits
      ckey$ = fstk$
      GOTO fform

```

slimits uses the field indirection operator **\$\$**:

```

slimits: REM fkey$$ gets the contents of the field named by fkey$
        SELECT LAST
        lstk$ = fkey$$
        SELECT FIRST
        fstk$ = fkey$$
        RETURN

```

The program returns to the next statement **fpage** after edit and delete functions. It too calls **slimits**, then executes **fform** to display from the first record on the current display page:

```

fpage:  REM Re-display page
        GOSUB slimits
        ckey$ = pstk$(1)

```

The program returns to **fform** after displaying the next or previous page, or after printing. The routine calls **spage** to display a page of records:

```

fform:  REM Display from ckey$
        GOSUB spage
        IF fst% THEN FORM
        fst% = 1

```

The **spage** subroutine displays up to the number of records given by **frow%** starting with the record given by **ckey\$**. It stores the key for each record displayed in the array **pstk\$**:

```

spage:  REM Displays page of records starting with ckey$
        REM Returns key array and number of records for page
        BLANK FORM
        SELECT KEY ckey$
        j% = 0
        FOR i% = 1 TO frow%
            UPDATE FORM ROW i%
            j% = j% + 1
            pstk$(i%) = fkey$$
            IF fkey$$ = lstk$ THEN i% = frow%
            IF i% < frow% THEN SELECT NEXT
        NEXT i%
        IF j% = frow% THEN lrec% = frow% ELSE lrec% = j%
        RETURN

```

The **fwait** routine immediately follows **fform**. This is the point at which the program waits for a pushbutton to be selected.

```

fwait:  REM Wait here for a push button
        mp% = 0
        SET STATUS ss13$
        WAIT MOUSE

```

Editing

```

REM Edit line
IF (mp% = 0 OR mp% > lrec%) THEN GOTO fwait
SET STATUS ss4$
SELECT FORM ROW mp%
fnd% = 0
ckey$ = fkey$$
MOUSE ON
IF mb% = cur% THEN ENTER Currency ROW mp%
IF mb% = cou% THEN ENTER Name ROW mp%,ffld%

```

If, during editing, the Name field in COUNTRY or the Currency field in CURRENCY is changed, the referential integrity check subroutine **sinteg** is called. The edited record can only be stored if **sinteg** returns 0 in fnd%.

```

IF (mb% = cou% AND Name.COUNTRY <> ckey$) OR (mb% = cur%
AND Currency.CURRENCY <> ckey$) THEN
  skey$ = fkey$$
  GOSUB sinteg
  IF fnd% = 0 THEN SELECT FORM ROW mp%:fkey$$ = skey$
END IF
IF fnd% = 0 THEN
  IF mb% = cur% THEN SET STATUS ss4$:ENTER Description ROW
  mp%,ffld% - 1
  REQUEST ms1$,"",127,b%
  IF b% = 1 THEN STORE
END IF
GOTO fpage

```

If the current file is COUNTRY, **sinteg** checks for the existence both of a CLIENTS record and of a STOCKS record with the corresponding country name. If the current file is CURRENCY, **sinteg** checks for the existence of a STOCKS record with the corresponding currency code. It displays messages as appropriate and, if any matching entries are found, returns a value 1 in fnd%.

```

sinteg: REM check not in use by clients or stocks
SET STATUS ss16$
IF mb% = cou% THEN
  FILE "clients"
  INDEX f$
  SELECT KEY ckey$
  IF FOUND ("") THEN REQUEST ms25$,ckey$ + " referenced by clients
file",100,a%:fnd% = 1
END IF
FILE "stocks"
INDEX f$
SELECT KEY ckey$

```

```
IF FOUND ("" ) THEN REQUEST ms25$,ckey$ + " referenced by stocks
file",100,a%:fnd% = 1
FILE f$
RETURN
```

New Record Entry

The Enter pushbutton accesses the new record entry routine **fenter**. This routine blanks the form and asks for input of a number of fields (ffld%) for the new record. If the entry is confirmed as correct, a new record is created. Further records can be entered on subsequent lines until the screen is full, at which point the routine blanks the form once more.

```
fenter: REM Enter new records
        BLANK FORM
        VIEW
        SET STATUS ss3$
        MOUSE ON
        mp% = 1
fenter2:
        ENTER (mp% - 1) * ffld% + 1,ffld%
        REQUEST ms1$,"",127,a%
        IF a% = 1 THEN STORE :mp% = mp% + 1
        REQUEST ms2$,"",130,a%
        IF a% = 0 THEN GOTO floop
        IF mp% <= frow% THEN GOTO fenter2 ELSE GOTO fenter
```

Record Deletion

The Delete pushbutton accesses the deletion routine **fdelete**. This routine waits for a mouse click on the line to be deleted, then calls **sinteg** to determine if the current record as selected by SELECT FORM ROW is referenced by other files. If it is not, the user is asked for confirmation with a REQUEST type 119, and the record is removed:

```
fdelete: REM Delete record

        SET STATUS ss12$
        WAIT MOUSE
        SELECT FORM ROW mp%
        ckey$ = fkey$$
        fnd% = 0
        GOSUB sinteg
        IF fnd% = 0 THEN
            REQUEST ms14$ + ckey$,"",119,a%
```

```
IF a% THEN
SELECT FORM ROW mp%
SELECT KEY ckey$
SELECT REMOVE
END IF
END IF
GOTO fpage
```

Next Page and Previous Page Display

The Next Page pushbutton accesses the routine **fnext**. This checks whether the last record in the current display is the last record on file. If so, it returns to **fwait**. If not, it makes the last displayed record the current record, selects the next record and branches to **fform** to re-display from this record.

```
fnext:  REM Next page
IF lstk$ = pstk$(lrec%) THEN GOTO fwait
SELECT KEY pstk$(lrec%)
SELECT NEXT
GOTO fcurr
```

The Prev Page pushbutton accesses the routine **fprev**. This checks if the first record in the current display is the first record on file. If so, it returns to **fwait**. If not, it makes the first displayed record the current record, selects as many previous records as there are lines on the screen and branches to **fform** to display from the now current record.

```
fprev:  REM Prev page
IF fstk$ = pstk$(1) THEN GOTO fwait
SELECT KEY pstk$(1)
FOR i% = 1 TO frow%
SELECT PREVIOUS
NEXT i%
fcurr:  REM Assign ckey
ckey$ = fkey$$
GOTO fform
```

The Daily Activities Menu

Currency Rates

This activity is the first item on the Daily activity menu and consists in entering new exchange rates for selected CURRENCY records. The date of the new rate is automatically set to the system date. The activity is controlled using a combination of the program STKDC and the form STKDC.

The STKDC form is designed for use with the STKDC program and includes the currency code, description and exchange rate for up to 12 records from the CURRENCY file, one record per line. Data is framed in the CURRENCY color of dark blue. There are three visible pushbuttons on the form, with associated commands as follows:

Exit	GOTO fexit
Next Page	GOTO fnext
Prev Page	GOTO fprev

There are also 12 hidden pushbuttons, one behind each data line. Each of these has an associated command assigning a line number to the variable mp%, e.g. mp% = 1.

Stock Prices

This activity is the second item on the Daily activity menu and consists in entering new asked and bid prices for selected STOCKS records. The date and time of the new prices are automatically set to the current system date and time. The activity is controlled using a combination of the program STKDC and the form STKDS.

The STKDS form is designed for use with the STKDC program and includes 10 lines of data, each line displaying the stock key, stock type, currency price asked and price bid from a STOCKS record. Data items are framed in the STOCKS color of light blue. Each data line has a hidden pushbutton associated with it. The visible pushbuttons are similar to those for STKDC.

Certificate Numbers

This activity is the third item on the Daily activity menu and consists in entering or editing certificate numbers for selected purchase TRANS records (type "b") where a certificate advice has not yet been sent. It is the only part of the system where multiple TRANS records can be edited on a single display. It is controlled using the combination of the form program STKDC and the form STKDN.

The STKDN form is designed for use with the STKDC program, and includes 10 lines of data, each line displaying the stock key, transaction reference, transaction

date, quantity and certificate number from a TRANS record. Data items are framed in the TRANS color of green. Each data line has a hidden pushbutton associated with it. The visible pushbuttons are similar to those for STKDC.

The Program STKDC

This program is similar in many respects to STKFO. It follows similar labeling conventions and uses similar algorithms for displaying pages of data. In the Daily Certificate Nos activity, display of TRANS records is subject to a filter, programmed at the beginning of STKDC as follows:

```
IF mb% = nos% THEN SELECT WHERE Trans_Type = "b" AND
  Cert_Sent_yn = "n"
```

STKDC contains no logic for creating or deleting records, only for editing them. The editing algorithm follows. Notice the use of ENTER...ROW with field names:

```
fwait:  REM Wait here for a push button
        mp% = 0
        SET STATUS ss13$
        WAIT MOUSE
        REM Edit line
        IF (mp% = 0 OR mp% > lrec%) THEN GOTO fwait
        SET STATUS ss4$
        SELECT FORM ROW mp%
        MOUSE ON
        SELECT CASE mb%
        CASE cur%
            ENTER USD Xrate ROW mp%
        CASE sto%
            ENTER Price Asked ROW mp% TO Price Bid ROW mp%
        CASE nos%
            ENTER Certificate_No ROW mp%
        END SELECT
        REQUEST ms1$, "", 127, a%
        IF a% THEN
            SELECT CASE mb%
            CASE cur%
                Date of Xrate.CURRENCY = TODAY
            CASE sto%
                Price Date.STOCKS = TODAY : Price Time.STOCKS = NOW
            END SELECT
            STORE
        ELSE
```

The next part of the routine restores the previous values in ROW mp% and re-displays the form in the event that the entry is incorrect:


```
IF k% = 1 THEN SELECT KEY pstk$(mp%) ELSE SELECT KEY  
pstk$(mp%)  
UPDATE FORM ROW mp%  
FORM  
END IF  
MOUSE OFF  
GOTO fwait
```

Cash Receipts and Payments

These activities are items 4 and 5 on the Daily menu, and consist in entering details of cash received from and paid to clients. Both activities are controlled using the combination of the form STKDR and the program STKDR. Each receipt or payment entered causes updating of the Cash Receipts balance in the relevant CLIENTS record, and the creation of a new record in the CASH file.

The Form STKDR

The two page form STKDR is designed for use with the program STKDR. The first page includes two pushbuttons as follows:

Exit	GOTO fexit
Enter	GOTO bhead

Clicking Enter causes the routine to increment the CTRL file batch number. The form displays the routine type (Payments/Receipts) using a pair of radio buttons. It displays the batch date and reference, and requests input of a batch total.

The second page of the form re-displays the routine type using radio buttons, together with the batch total, batch date and reference. It includes a block for the display of up to 10 line items, each consisting of the transaction date, time, transaction reference, client reference, client net due, cash reference and cash amount. The client net due, Amount Due.CLIENTS, is retrieved following input of a valid client reference through the form link between Client.CASH and Customer_Ref.CLIENTS. At the foot of the form are displayed the total cash amount for the current page, and the total entered for the batch. This page includes pushbuttons as follows:

Exit	GOTO fexit
Enter	GOTO fenter
Delete	GOTO fdelete
Next Page	GOTO fnext
Prev Page	GOTO fprev

Behind each data line is a hidden pushbutton with an associated command assigning the line number to the variable mp%.

The Program STKDR

```

REM   Enter Cash Receipts and Payments
REM   Last updated Jan 03, 90
ON ERROR GOTO ferror
REM   STKDR
REM   Enter Cash etc.
REM   Last updated etc.

```

```
stkdr: REM Set parameters
```

```
  f$ = "cash":fkey$ = "Trans_Key":fmd$ = "stkdr":ffld% = 7:frow% = 10
```

The following command assigns a value to btype\$ of "r" for Receipts or "p" for Payments, depending upon the menu column variable mb%:

```

IF mb% = 4 THEN btype$ = "r" ELSE btype$ = "p"
bdate$ = DATE$ ( TODAY , "mmm dd,yy"):bdate% = TODAY
OPEN FILE f$
INDEX fkey$
OPEN FORM fmd$

```

The following routine awaits a mouse click on page one of the STKDR form:

```

bwait: REM Enter batch header or exit
WAIT MOUSE
GOTO bwait

```

Clicking on the Enter pushbutton causes batch.CTRL to be incremented:

```

bhead: REM Batch header
FILE "ctrl"
batch.CTRL = batch.CTRL + 1
STORE
bref% = batch.CTRL
SET STATUS ss20$

```

The following section gets input of the batch total:

```

bhead2:REM Batch total
FILE f$
ENTER 7
IF Zero <= 0 THEN REQUEST ms27$,"",2,a%:GOTO bhead2
REQUEST ms1$,"",130,a%
IF a% = 0 THEN GOTO bhead2
bttl% = Zero:Zero = 0:ftl% = 0:tref% = 1

```

The next section displays the second page of the form:

```

fenter: REM Enter Records from line 1
BLANK FORM

```

FORM 2

SET STATUS ss3\$

mp% = 1:ttl% = 0

Starting with row number mp% = 1 and the transaction reference tref% for this batch = 1, the following section requests input of line details up to Cash_ref:

fenter2: REM Enter a record

BLANK

Trans_ref.CASH = tref%

UPDATE FORM ROW mp%

ENTER Trans_date ROW mp% TO Cash_ref ROW mp%

The next section requests input of the transaction amount, which is verified a positive. If entry is confirmed as correct, the running page and batch totals are incremented by the entered amount, the print flag set to "n", and a new record written to CASH. The Cash_Receipts amount in the CLIENTS record is updated by the subroutine **supdate**:

fenter3: REM Enter cash amount

ENTER Amount.CASH ROW mp%

b% = 0

IF Amount.CASH > 0 THEN

REQUEST ms1\$, "", 127, a%

IF a% THEN

amt% = Amount.CASH:ttl% = ttl% + amt%:ftl% = ftl% + amt%

tref% = tref% + 1:mp% = mp% + 1

Batch_ref.CASH = bref%:Batch_date.CASH = bdate%

Printed_yn.CASH = "n":Trans_Type.CASH = btype\$

STORE

GOSUB supdate

FORM

END IF

ELSE

REQUEST ms27\$, "", 100, a%:b% = 1

END IF

IF b% THEN GOTO fenter3

REQUEST ms2\$, "", 130, b%

IF b% = 0 THEN GOTO fcheck

IF mp% > frow% THEN GOTO fenter ELSE GOTO fenter2

When the user indicates no further entries, the running batch total is checked against the batch total entered at the beginning of the program. If the totals agree, the option is given to edit the batch, otherwise the user is put into edit mode, and asked to click on a line to edit:

fcheck: REM Check batch total

```

IF ftl% = bttl% THEN
  REQUEST ms28$,ms29$,130,b%
ELSE
  REQUEST ms26$,ss13$,100,a%:b% = 1
END IF
IF b% = 0 GOTO fexit

```

Edit mode re-displays CASH records corresponding to the current batch number, using similar display logic to STKFO:

```

floop:  REM Re-display from first record
        BLANK FORM
        SELECT WHERE Batch_ref.CASH = bref%
        GOSUB slimits
        ckey% = fstk%
fform:  REM Display from ckey
        GOSUB spage
        FORM 2

```

The following section waits for a pushbutton selection. This might be one of the visible pushbuttons Exit, Enter, Delete, Next Page or Prev page, or one of the hidden line pushbuttons to indicate the line to be edited:

```

fwait:  REM Wait here for a push button
        mp% = 0
        SET STATUS ss13$
        WAIT MOUSE

```

The next section of STKDR deals with line editing based on a line number mp% in the range 1 to lrec%. The editable fields in the transaction line are the cash reference and cash amount. Other program sections **fdelete**, **fnext** and **fprev** deal with line deletion and next or previous page display. These sections are similar in logic to the equivalent sections in STKFO. The section **fexit** checks the running batch total against the entered batch total and if not equal, returns to **fwait**:

```

fexit:  REM Exit
        IF ftl% <> bttl% THEN REQUEST ms26$,ss13$,100,a%:GOTO fwait
        FILE "ctrl"
        CHAIN "menu"

```

The Document Production Menu

Certificate Advices

Certificate advices are sent to all clients with associated purchase transactions (type "b") where a stock certificate has now been received and its number entered into the system using either File Transactions or Daily Certificate Nos.

Certificate advices are actually personalized letters produced using the MERGE command with the form letter LECERTS.SBT in conjunction with the temporary Superbase file CERTS. The entire production process is under the control of the **m31** section of MENU (see below).

Contract Notes

A contract note is printed for each sale or purchase transaction. Contract notes are personalized letters produced using the MERGE command with the form letter LEINVS.SBT in conjunction with the temporary Superbase file INVS. The production process is also under the control of the **m31** section of MENU.

The MENU Program Section m31

The first section of **m31** displays the appropriate warning message, giving the user the option to return to the main menu:

```
m31:  REM Documents Certificate advices, Contract notes
      SELECT CASE mb%
      CASE 1
        REQUEST ms311$,ms312$,140,a%
      CASE 2
        REQUEST ms321$,ms322$,140,a%
      END SELECT
      IF a% = 0 THEN RETURN
```

The first step in producing certificate advices is to create a new temporary file CERTS containing data from the qualifying TRANS records together with ancillary data from the CLIENTS and STOCKS files. This new file will then form the basis of a mailmerge operation. The reason for the new file is that we want to include data from a number of files in the certificate advice. However MERGE will only function with a single file.

```
MOUSE OFF
SET STATUS ss8$
IF mb% = 1 THEN
  REM Certificate advices
  docf$ = "certs":letf$ = "lecerts"
  IF EXISTS ("certs.sbf") THEN OPEN FILE docf$:REMOVE FILE docf$
```

```

SELECT
Customer_Ref.TRANS,Quantity.TRANS,Certificate_No.TRANS,
Title.CLIENTS,First
name.CLIENTS,Lastname.CLIENTS,Company.CLIENTS,
Street.CLIENTS,Address.CLIENTS,City.CLIENTS,State.CLIENTS,Zip_Co
de.CLIENTS, Country.CLIENTS,Company Name.STOCKS,Stock_Type

```

The WHERE clause sets up the links between the TRANS and the CLIENTS and STOCKS files and imposes three conditions: the TRANS record is a purchase, the Certificate_No field is other than null and no certificate advice has yet been produced:

```

WHERE Customer_Ref.TRANS = Customer_Ref.CLIENTS AND
Stock_Key.TRANS = Stock_Key.STOCKS AND Trans_Type.TRANS =
"b" AND Certificate_No.TRANS > "" AND Cert_Sent_yn.TRANS = "n"
ORDER Stock_Key.TRANS,Certificate_No.TRANS
TO FILE docf$
END SELECT

```

For contract notes, the temporary file is called INVS. Notice the use of the STR\$ function in the following SELECT statement. It enables correct formatting of numeric data in the temporary file prior to MERGE. Also notice the use of ABS. The contract note will indicate whether the transaction is a sale or a purchase by means of the Trans_Desc. The ABS function eliminates the sign from the transaction amount.

```

ELSE IF mb% = 2 THEN
  REM Contract notes
  docf$ = "invs":letf$ = "leinvs"
  IF EXISTS ("invs.sbf") THEN OPEN FILE docf$:REMOVE FILE docf$
  SELECT Stock_Key.TRANS,Customer_Ref.TRANS,Transaction ref,
Trans_Type,Trans_Desc,Trans_Date,Settlement_Date, STR$
(Quantity.TRANS,6,0) AS "QTY", STR$ (Stock_Price.TRANS,6,2) AS
"PR",Currency.TRANS,USD Xrate.TRANS,STR$ ( ABS (USD
Value.TRANS),6,2) AS "USDV", (Comm_Value) AS "Com", STR$ ( ABS
(USD Value.TRANS + Comm_Value),6,2) AS "TV",&24(Client.CLIENTS)
AS "Client",
Company.CLIENTS,Street.CLIENTS,Address.CLIENTS,City.CLIENTS,Sta
te.CLIENTS, Zip_Code.CLIENTS,Country.CLIENTS,Company
Name.STOCKS,Stock_Type,Exchange

```

The WHERE clause establishes the links between the TRANS and the CLIENTS and STOCKS files, and select only those TRANS records where no contract note has yet been produced:

```

WHERE Customer_Ref.TRANS = Customer_Ref.CLIENTS AND
Stock_Key.TRANS = Stock_Key.STOCKS AND
Invoice_Sent_yn.TRANS = "n"
ORDER Customer_Ref.TRANS,Transaction ref

```

```

        TO FILE docf$
    END SELECT
END IF

```

The next section opens the temporary file docf\$, asks whether output is to be directed to the screen or the printer, loads the form letter letf\$ and produces a merge document for each temporary file record:

```

        OPEN FILE docf$
        GOSUB sdorp:REM display or print
        IF a% = 0 THEN REMOVE FILE docf$:RETURN
m312:   REM Display or print merge documents
        LOAD TEXT letf$:SELECT FIRST
        WHILE NOT EOF (docf$)
            IF p$ = "p" THEN MERGE TEXT letf$
            IF p$ = "d" THEN ? TEXT MERGE
            WAIT FOR 2
        SELECT NEXT
    WEND
    b% = 0

```

The following REQUEST asks whether output has been correctly printed.

```

    REQUEST ms4$,"",130,a%

```

If output is complete, the routine updates the flags in CTRL to cause menu items associated with amending the TRANS file to be re- displayed as grayed and the relevant Update menu option to be re- displayed as active:

```

    IF a% THEN
        REM Update CTRL file and re-set menus grayed=0 active=1
        FILE "ctrl"
        IF mb% = 1 THEN
            tran.CTRL = 0:cent.CTRL = 0:cert.CTRL = 0:ceru.CTRL = 1
            MENU 1,5,tran.CTRL,"&Transactions"
            MENU 2,3,cent.CTRL,"Certificate &Nos"
            MENU 3,1,cert.CTRL,"&Certificates"
            MENU 4,3,ceru.CTRL,"&Certs Sent"
            ELSE IF mb% = 2 THEN
                tran.CTRL = 0:cont.CTRL = 0:conu.CTRL = 1
                MENU 1,5,tran.CTRL,"&Transactions"
                MENU 3,2,cont.CTRL,"C&ontract Notes"
                MENU 4,4,conu.CTRL,"&Notes Sent"
            END IF
        STORE
    ELSE
        REQUEST ms3$,"",135,b%
        IF b% = 0 THEN REMOVE FILE docf$
    END IF

```

```
IF b% = 1 GOTO m312
RETURN
```

Statements

Client Statements are used to display and print client details together with details of the transactions entered into on behalf of the client. Statements are controlled using the form STKOS, together with the MENU program section **stkf**.

The statement form STKOS contains selected fields from the CLIENTS file, including the virtual name field Client.CLIENTS and other fields making up the address. There is also a one-to-many transaction block consisting of 8 lines of data from the TRANS file, each line displaying details from a separate TRANS record. The TRANS records displayed are those where the link applies, namely where Customer_Ref.CLIENTS is equal to Customer_Ref.TRANS.

Three calculations are included on STKOS, namely the totals for the US\$ Value, Commission and Total Due columns. The following pushbuttons are also included:

Exit	GOTO stexit
Next Page	SELECT FORM NEXT PAGE:VIEW
Prev Page	SELECT FORM PREVIOUS PAGE:VIEW
Index	GOSUB index
Browse	GOSUB spanel
Print	GOSUB sprint

The Update Menu

New Highs

This activity is first option on the Update menu, and is carried out by the **m41** section of MENU. This routine processes the STOCKS file, updating the current year high price and date as appropriate. The routine first displays a dialog describing the update that has been selected, and gives the option to continue or to cancel:

```
m41:  REM Update New Highs
      REQUEST ms421$,"",140,a%
      IF a% THEN
        MOUSE OFF
        SET STATUS ss19$
```

The following UPDATE WHERE command says if the current Price Middle is higher than the previous Year High price or if the Year High is last year's, then make the Year High price and the Date of High the current Middle price and Date:


```

      UPDATE Year_High.STOCKS = Price Middle.STOCKS:Date of
High.STOCKS = Price Date.STOCKS
      WHERE Price Middle.STOCKS >= Year_High.STOCKS OR YEAR
      (Price Date.STOCKS) > YEAR (Date of High.STOCKS)
      END UPDATE
    END IF
  RETURN

```

New Lows

This activity in second option on the Update menu, and is carried out by the **m42** section of MENU. This routine processes the STOCKS file, updating the current year low price and date as appropriate. This routine is a mirror image of the New Highs Update **m41** described above.

Certificates Sent

This activity in third option on the Update menu, and is carried out by the **m43** section of MENU. This routine updates the Certificates Sent flag in the TRANS file to indicate that certificates have been produced.

```

m43:  REM Update Certs Sent
      REQUEST ms431$,ms432$,140,a%
      IF a% THEN
        MOUSE OFF
        SET STATUS ss19$

```

The UPDATE WHERE clause uses the same criteria as were used by the certificate advice production routine **m31**:

```

      UPDATE Cert_Sent_YN.TRANS = "y"
      WHERE Certificate_No.TRANS>" AND Cert_Sent_YN.TRANS = "n"
      END UPDATE
      REM Update CTRL file and re-set menus grayed=0 active=1

```

We can now re-activate the File Transactions and Daily Certificate Nos menu options, as well as the Documents Certificates option. The Update option is grayed until the next document production cycle.

```

      FILE "ctrl"
      tran.CTRL = 1:cent.CTRL = 1:cert.CTRL = 1:ceru.CTRL = 0
      MENU 1,5,tran.CTRL,"&Transactions"
      MENU 2,3,cent.CTRL,"Certificate &Nos"
      MENU 3,1,cert.CTRL,"&Certificates"
      MENU 4,3,ceru.CTRL,"&Certs Sent"
      STORE
    END IF

```

RETURN

Contract Notes Sent

This activity is fourth option on the Update menu, and is carried out by the **m44** section of MENU. This routine updates the Contract Sent flag in the TRANS file to indicate that contract notes have been produced.

```
m44:  REM Update Contract Notes Sent
      REQUEST ms431$,ms442$,140,a%
      IF a% THEN
        MOUSE OFF
        SET STATUS SS19$
```

The UPDATE WHERE clause uses the same criteria as were used by the contract note production routine **m33**:

```
      UPDATE Invoice_Sent_yn.TRANS = "y" WHERE
      Invoice_Sent_yn.TRANS = "n"
      END UPDATE
      REM Update CTRL file and re-set menus grayed=0 active=1
```

We can now re-activate the File Transactions and the Documents Contract Notes options. The update option is grayed until the next production cycle.

```
      FILE "ctrl"
      tran.CTRL = 1:cont.CTRL = 1:conu.CTRL = 0
      MENU 1,5,tran.CTRL,"&Transactions"
      MENU 3,3,cont.CTRL,"C&ontract Notes"
      MENU 4,4,conu.CTRL,"&Notes Sent"
      STORE
    END IF
  RETURN
```

23 PROGRAMMING IDEAS

Customizing the Superbase Environment

Category: System Management

Covers...

Using the default start program START.SBP.

Using the DML commands PANEL OFF, BREAK OFF, CLS, DIRECTORY, WAIT, QUIT.

The Problem

When you are preparing an application, you may want to put your users straight into an application environment, rather than giving them access to all of the Superbase functions from the native menu. You will need to ensure that at least some of the following things happen automatically each time a user calls up Superbase:

- The application directory is made the current directory
- The application files and forms are opened
- Default function key settings are loaded
- A menu of activities is displayed
- The display is customized by removing the browsing panel, clearing the screen, displaying a banner, etc.

The Solution

You can do all these things and more with a start program. Each time Superbase is loaded, it looks for a program called START.SBP in its startup directory. If found, this program is automatically loaded and executed. Using a start program allows you to customize the Superbase environment to the needs of a particular application or machine. In a networked environment you can use a separate start program for each user (see 5-34).

The following example start program illustrates in outline how this might be done. Notice the use of a simple character-oriented menu command strategy and the use of a type 23 REQUEST (no echo) for entering a password:

```
REM Start.SBP
PANEL OFF
BREAK OFF
CLS
```

```
DIRECTORY "DH0:MYDATA"
```

```
? "Superbase Startup Options. Please choose an activity"
```

```
? :?
```

```
? "1. Invoicing"
```

```
? "2. Stock Control"
```

```
? "3. Mail Merge"
```

```
? "4. Utilities Menu"
```

```
?
```

```
? "5. Native Superbase"
```

```
? "6. Leave Superbase and return to DOS"
```

```
loop:
```

```
WAIT a$
```

```
a% = VAL (a%)
```

```
IF a% < 1 OR a% > 6 THEN GOTO loop
```

```
ON a% GOTO s1, s2, s3, s4, s5, s6
```

```
s1:
```

```
? "Running the Invoicing application"
```

```
CHAIN "INVOICES"
```

```
s2:
```

```
? "Running the Stock Control Program"
```

```
CHAIN "STOCKS"
```

```
s3:
```

```
OPEN FILE "Merge"
```

```
MERGE TEXT "Letter" WHERE ASK
```

```
GOTO loop
```

```
s4:
```

```
CHAIN "UTILS"
```

```
s5:
```

```
REQUEST "Please enter Password", "For native Superbase", 23, p%, p$
```

```
IF p$ = "AccessCode" AND a% = 1 THEN END
```

```
GOTO loop
```

```
s6:
```

```
REQUEST "Exit Superbase - are you sure?", "", 1, p%
```

```
IF p% = 1 THEN QUIT
```

```
GOTO loop:
```

Adding Password Protection to Activities

Category: System Management

Covers...

Using REQUEST to enter non-echoed passwords.

Protecting access to user-defined passwords.

The Problem

Databases are often used to store confidential information. When the machine is situated in an open office, or the information is stored on a networked system, then security becomes an important issue.

Basic security within Superbase is implemented in two ways. Firstly by password-restricting access to data files, and secondly through the encryption of user DML programs. Passwords specified at the file definition level enable the system manager to control who is allowed to read, modify, add or delete records. The DML PROTECT command prevents users from reading or modifying an application's control programs.

You may however want to allow some users access to some but not all records within a data file, or to control access to programmed activities within an application rather than controlling access to the files themselves.

The Solution

You can achieve additional levels of access security through the use of one of the user-defined REQUEST types, type 23. This request allows the user to type information without it being echoed back to the screen. It is thus ideal for entering passwords.

As the system manager, you can implement password protection at any number of strategic points throughout an application. On first running Superbase, the START program could prompt a password to determine whether a user has the access rights to run an application. Further passwords could be prompted when the user opens a form. Using the Command object type, the password request could even be called from a form itself.

You can embed SELECT statements within file maintenance and enquiry programs so as to provide access by means of passwords to different subsets of complete files. One password gets the entire client file, another only the accounts in territory x, another only the accounts where the company name starts with the letters A through E, and so on.

Clearly for passwords to be effective they cannot be accessible to unauthorised persons. The most obvious place for an intruder to look for a list of passwords is in the program that performs the password checking. As an elementary precaution,

this program must be encrypted so that it cannot be listed. But suppose your application is in an environment where passwords need to be changed, added or removed fairly regularly. The protected program containing the list of passwords cannot be edited, so an unprotected copy must be kept in a safe place. This is in itself a security risk which must be carefully considered.

An alternative approach is to store your passwords in a Superbase file and simply look them up as needed. Since Superbase does not encrypt text data within a file in any way, you would then have to invent your own method of encryption. This approach allows you to change, add or remove passwords simply by amending records in the file. Naturally access to the file should be password-protected. The following program segment uses this latter approach to check a password entered by a user.

```
REQUEST "Please type your password","and press <RETURN>", 23,
a%, a$
IF a$ = "" OR a% = 0 THEN END
GOSUB encrypt :REM Encryption algorithms tend to be quite long
REM and so none is included. Assume it returns enc$
LOOKUP ( enc$,Code.PASSWORDS)
IF NOT FOUND THEN
    ? "Sorry you have typed an invalid password"
END
END IF
REM Determine user access
access% = valid.PASSWORDS
ON access% GOTO route1, route2, route3
```

Unattended File Transfer

Category: Communications

Covers...

Using DML communications commands to transfer files.

The Problem

Separate computers may be in use at different locations to perform related tasks. For example one computer may be used at a central warehouse location for inventory control, while others are in use at separate sales locations for entering orders. As it is not practicable to connect them in a Local Area Network, the only method of communication is via modem.

Systems can be devised whereby for example the details of orders paced during the day at each sales location can be used to update the inventory availability information which forms part of the central inventory system. New free stock levels can then be used on the following day by each sales location.

The problem is how to pass the information back and forth between the various locations without having to constantly convert files from one format to another and without constant operator supervision.

The Solution

The telecommunications module incorporated within Superbase facilitates the transfer of files between remote computers. This can be accomplished either through the native menu system, which is very easy but requires some supervision, or through use of DML commands, in which case it can be programmed to proceed automatically.

Communications commands within DML include the OPEN COMMS, CLOSE COMMS, COMMS FILE GET and COMMS FILE ? commands.

The following example is of interest in a situation in which both transmitting and receiving computers are using Superbase. Two routines are given: one to transmit the files and the other to receive them.

The transmitting routine assumes that there is a pre-defined list of files to be transmitted. This list is stored in the variable fil\$, and consists of a normal DOS path followed by the filename. For multiple file transfers, wild card characters can be used in commands.

The transmitting routine opens a comms channel and specifies the parameters to be used for the modem. In the example, the parameters are 2400 baud in WXModem protocol with Auto headers on (enabling multiple file transfers). The receiving routine sets up exactly the same parameters, except that the transmit/receive parameter is set to receive.

Opening the comms channel does not initiate modem dialling. This is done via a separate subroutine **dial**. The Hayes standard modem command to tone dial a number is ATDT followed by the number. The program then waits for the connection to be established with the receiver. An option is included here for a key press to terminate the process. Note that, as an alternative to this, the COMMS FILE ? command would automatically send an initialization string and dial string if one was specified in OPEN COMMS. An error trap would then have to be included with the COMMS FILE ? to deal with a connect failure.

```
REM Comms file transfer - Transmit
OPEN COMMS USING 1,2400,0,1,8,1,1,1,1
GOSUB dial
COMMS FILE ? fil$
CLOSE COMMS
END
```

dial:

```
COMMS ? "ATDT90001112222"
WAIT COMMS OR KEY
GET a$
IF a$ <> "" THEN
    CLOSE COMMS
    REQUEST "Comms Transfer failed", "", 0, a%
END
END IF
RETURN
```

```
REM Comms file transfer - Receive
OPEN COMMS USING 1,2400,0,1,8,0,1,1,1
GOSUB connect
COMMS FILE GET fil$
CLOSE COMMS
END
```

connect:

```
WAIT COMMS OR KEY
GET a$
IF a$ <> "" THEN
    CLOSE COMMS
    REQUEST "Comms Transfer failed", "", 0, a%
END
END IF
RETURN
```


Sending and Receiving Text

Category: Communications

Covers...

Using COMMS INPUT to communicate text a line at a time.

Using COMMS GET to communicate text a character at a time.

The Problem

The telecommunications commands incorporated within DML considerably extend the range of activities that can be programmed and incorporated within an application. For example, a program could be written to automatically dial up a comms-based stock price information service at pre-defined intervals, download current prices into a database, calculate price fluctuations and initiate appropriate warning actions.

The problem is that the facilities provided by Superbase appear to be oriented towards file transfer, but there are many instances where files are not involved. You may want to use Superbase as a simple terminal for accessing bulletin boards and online conferencing systems. In general, you want to be able to send and receive lines of text to and from the screen as well as files to and from the disk.

The Solution

The following example shows you how to set about the problem of communicating text, either a line or a character at a time. The example includes the BREAK OFF command. This is necessary in order for a space to be typed without Superbase interpreting it as a keyboard request to pause execution of the program.

Two slightly different routines are included: one uses COMMS INPUT for a line based application – the message will not appear on the screen until a whole line has been sent. The other uses COMMS GET for a character based application – each character is individually received and printed to the screen. The character-based routine will clearly be much slower if you are receiving a large number of long lines.

```
SET PAGING ON :CLS
? @23,7"SIMPLE COMMUNICATIONS DEMO PROGRAM"
? @32,10"CTRL-C to EXIT"
SET PAGING OFF :REM force screen scrolling and allow get space
BREAK OFF
OPEN COMMS USING 0,1200,0,1,8:REM Use port 0 at 1200 baud, no
parity
REQUEST "Do you want screen echo","",21,dx%:REM 1 for screen echo
GOTO lxferr
```

```
REM *** The lxfer section uses COMMS INPUT
REM *** The cxfer section uses COMMS GET
```

cxfer:

```
GET t$:IF t$ = CHR$ (3) THEN
  CLOSE COMMS
  SET PAGING ON
  END
END IF
IF t$ = CHR$ (13) THEN t$ = t$ + CHR$ (10)
IF t$ <> "" THEN
  IF dx% THEN ? t$;:REM Copy to screen if required
  COMMS ? t$
END IF
COMMS GET TEXT r$:IF r$ = "" THEN GOTO cxfer
? r$;:GOTO cxfer
```

lxfer:

```
GET t$:IF t$ = CHR$ (3) THEN
  CLOSE COMMS
  SET PAGING ON
  END
END IF
IF t$ = CHR$ (13) THEN t$ = t$ + CHR$ (10)
IF t$ <> "" THEN
  IF dx% THEN ? t$;:REM copy to screen if required
  COMMS ? t$
END IF
COMMS INPUT TEXT r$:IF r$ = "" THEN GOTO lxfer
? r$;:GOTO lxfer
```

Importing Data from Non-ASCII Files

Category: Import

Covers...

Using OPEN FOR INPUT and INPUT to import non-ASCII data.

Using the SER function to give a file a record number sequence.

The Problem

There are many situations in which you want to convert data stored in other file formats into Superbase format. The standard Import facility can cope with many of the more popular file formats as well as with standard ASCII. (see Chapter 21 Import, Volume 1)

However, many files that might appear at first sight to be fixed length ASCII do in fact contain embedded nulls or control characters. These characters will cause Import to malfunction. You can check whether the file you want to Import contains such characters by using a public domain file-debugging utility, in which control characters are represented by a period.

Examples of non-ASCII files are fixed length files downloaded from a mainframe containing embedded carriage return and/or linefeed characters (ASCII codes 13 and 10), or files maintained by other DOS applications and containing embedded nulls.

The Solution

Such non-ASCII files can be merged into Superbase files under DML control using the OPEN FOR INPUT and INPUT commands. An important point to remember when attempting this is that Superbase treats a null (ASCII code 00) as a string terminator.

The following example deals with the importing of a General Ledger chart of accounts structure from a popular accounting package. In this package the account details, for example the account name, the account type and the related major account code, are held in a file called DA3-F00.DAT. This file has a fixed record length of 79 characters. Deleted records and unused fields are filled with nulls.

The account codes themselves are held in a separate file called DA3-F00.KEY. This file can be discovered by inspection to have a fixed record length of 8 characters. For valid records the first character is a 1 and for deleted records the first character is a zero (as distinct from a null). Any other value in the first character position indicates end of file. The eighth character is always a null.

The object is to merge not just one but both of these files into a new Superbase file. The relevant part of the new Superbase file definition DEAGL is as follows:

```

Number      ;TXT REQ      IXU      ;6 U ;0 ;9;
Name        ;TXT          ;20 C    ;1 ;11 ;
Type_GD     ;TXT VAL REQ  ;1 U     ;2 ;8  ;>
Type_GD.DEAGL LIKE "g" OR Type_GD.DEAGL LIKE "d"
Level_12345 ;NUM VAL      ;9.      ;3 ;4  ;>
Level_12345.DEAGL > 0 AND Level_12345.DEAGL < 6
General Account;TXT          ;6 U     ;4 ;0  ;
Class       ;TXT          ;1        ;2 ;40  ;
Recno       ;NUM CON RDO IXD;9999999. ;15 ;10 ;>
SER ("deagl")

```

The example below proceeds in two steps. The first step reads through DA3-F00.KEY sequentially, creating a new record in DEAGL for each valid account key found in DA3-F00.KEY. The second step reads through DA3-F00.DAT sequentially, writing additional account information to the corresponding record in DEAGL.

The purpose of the field Recno.DEAGL is to enable this second pass to write to DEAGL records in the same sequence as they were created. Recno.DEAGL is defined as SER("DEAGL"). Each new DEAGL record will be given the next Recno in an incrementing sequence. If these records are then accessed using SELECT FIRST, SELECT NEXT with Recno.DEAGL as the index, they will be accessed in the same sequence as they were created.

```

REM Read Chart of Accounts data into DATEASY G/L Account file
OPEN FILE "DEAGL"
OPEN "DA3-F00.KEY" FOR INPUT

```

OPEN FOR INPUT automatically positions the file pointer at the first character position.

```

I1: INPUT &8,f1$
  IF LEFT$(f1$,1) = "0" THEN GOTO I1
  IF LEFT$(f1$,1) <> "1" THEN GOTO I2

```

0 in the first position indicates a deleted record and anything other than 1 (ASCII code 31) the end of file. Since recno.DEAGL is defined as a constant, the next value for SER("DEAGL") will automatically be written with the record.

```

BLANK FILE "DEAGL"
Number.DEAGL = MID$(f1$,2,6)
STORE :GOTO I1

```

The next section opens the non-ASCII data file and makes the first record created above the current record. DA3-F00.DAT is automatically at the first character position.

```
I2: OPEN "DA3-F00.DAT" FOR INPUT
    INDEX Recno.DEAGL:SELECT FIRST
```

The relevant data in the DA3-F00.DAT record is spread over the first 30 bytes. For present purposes we are not interested in the contents of bytes 31 to 79. We use several INPUT statements to read discrete chunks of data into strings, making sure that nulls are properly dealt with. A null in the first position indicates a deleted record. We therefore include a test to ignore the record if the length of the first string is zero. The 21st byte always contains a null. If the first three statements below were replaced by INPUT &30, fn\$, the fn\$ string would terminate at byte 21 and the data in bytes 22 to 30 would not be read. So we "trap" the null at byte 21 in f2\$, and the data that follows in f3\$.

```
I3: INPUT &20,f1$
    INPUT &1,f2$
    INPUT &9,f3$
    INPUT &49,f4$
    IF LEN (f1$) = 0 THEN GOTO I3
```

```
I4: Name.DEAGL = f1$: Class.DEAGL = LEFT$ (f3$,1)
    Level_12345.DEAGL = ASC ( MID$ (f3$,2,1))
    Type_GD.DEAGL = MID$ (f3$,3,1)
    General Account.DEAGL = MID$ (f3$,4,6)
    STORE :SELECT NEXT
    IF NOT EOF ("DEAGL") THEN GOTO I3 ELSE END
```

Formatting Output for Desktop Publishing

Category: Output

Covers...

Using the DML ? print command and OPEN FOR OUTPUT to format output.

Using OPEN FOR INPUT and INPUT LINE to read a file.

The Problem

Desktop Publishing (DTP) packages generally include import-type facilities for reading in text created by popular word processing (WP) products. Text is read in under the control of a pre-defined style-sheet, which "flows" it into a particular format, font style, etc. Once the appropriate style sheets have been created, presentation quality documents can easily be produced from word processed text without further manual formatting.

Many users want to be able to do the same thing with output from the database, namely to produce it to today's presentation standards, and to include it within desktop published documents and reports.

Using Superbase's standard output facilities it is possible to enhance presentation quality. Forms provide a means of incorporating multiple fonts, pitches and styles, and of combining data with edited text and graphics. Longer reports can be defined to include styles, headings and footings, calculations and so on. But the tools provided by Superbase are not intended to perform or replace conventional DTP page layout functions. And even when you use the standard output facilities to create ASCII disk files rather than screen or printer output, such files are not ready for DTP in the same way as WP files.

The Solution

To produce output from Superbase ready to be flowed into a style-sheet you basically have to understand the way the WP formats its output, then you can use DML print facilities to follow the same rules. You will then be able to fool the DTP into thinking it is getting its input from a WP.

An alternative of course is to output ASCII files to disk, then actually read them into a suitable WP, outputting them again in WP format. But if you are intending to create a close interface between Superbase and DTP, these are the major points to bear in mind:

Carriage returns and line feeds

WP output tends to treat the combination of a carriage return (CHR\$(13)) and line feed (CHR\$(10)) as a paragraph marker. Superbase inserts both when it executes a print statement, hence you should terminate all DML print statements other than paragraph endings with a semi colon (;). If the WP you are emulating uses a single

line feed character as the paragraph marker, then you can issue a paragraph marker with the DML command:

```
? CHR$(10); (notice the semi colon).
```

Space characters and tabs

Multiple space characters cause confusion. It is better to use the tab character CHR\$(9) instead. You must also ensure that tabs are correctly positioned in your DTP style-sheet. Correct positioning means that the data or text following a tab character in your print statement is never wider than the number of characters from the current to the next tab position in the style-sheet.

In the following example, suppose your style sheet has tabs at columns 7, 15, 27, 30 and 45, then to print fielda.FILE at column 7 and fieldb.FILE at column 30, you would issue the following DML command:

```
? CHR$(9) fielda.FILE; CHR$(9) CHR$(9) CHR$(9) fieldb.FILE;
```

In this example fielda.FILE must be less than 7 characters long, otherwise the first part of the print statement would leave the character pointer beyond the second tab and there would be one too many occurrences of CHR\$(9) in the command.

Use of TRIM\$

To avoid outputting multiple spaces, you can use the TRIM\$ function on field data. Then if you are appending multiple fields you will need to insert a single space between each field. Also remember when appending fields that in these circumstances Superbase can only handle strings of up to 4000 characters.

In the following example, suppose you are outputting these details from a number of records:

- text descriptions from fielda.FILE and fieldb.FILE
- date from fieldc.FILE
- text description from fieldd.FILE

Rather than concatenating all fields in a single string or print statement:

```
? TRIM$ (fielda.FILE) + " " + TRIM$ (fieldb.FILE)+ " " + "Dated: " +  
DATE$(fieldc.FILE,"0d/mm/yy") + " " + TRIM$(fieldd.FILE);
```

It is safer to issue separate print statements for each component:

```
? (fielda.FILE) + " ";  
? TRIM$ (fieldb.FILE)+ " ";  
? "Dated: " + DATE$(fieldc.FILE,"0d/mm/yy") + " ";  
? TRIM$(fieldd.FILE);
```

Note that if your text fields contain carriage returns, these should be eliminated before output. A carriage return in a multi-line Superbase text field consists of the two characters CHR\$(13) and CHR\$(10).

Use of String Variables for Formatting

Rather than repeat tab characters and line feeds throughout your program you may want to assign them to string variables, for example:

```
t1$=CHR$(9): t2$=CHR$(10)
```

You can incorporate DTP body-style tags within your text, and assign these to string variables also:

```
tag1$="<newrec>": tag2$="<descript>"
```

Finally, if you want to utilize some WP attribute that is recognized by the DTP, for example italics, the trick is simply to discover the special ASCII codes used by the WP and include them in your output. You can assign these to strings also:

```
italon$=CHR$(192)+CHR$(195): italloff$=CHR$(192)+CHR$(196)
```

Header Information

Some word processors carry formatting information about the file in a header. Again it is possible to fool the DTP by copying across the header information from some other WP file and using it as the header for your own output. In the example below the file TESTWP.XX has been written by the WP to store a single line of text as follows:

"This is plain and the word *italic* is italicized"

The file TESTWP.XX is found upon examination by a disk utility to contain 119 lines of header information followed by a line:

This is plain and the word [192][195]italic[192][196] is italicized

The following is an example of all of the above points. It aims to create a file DTPOUT.XX for automatic DTP formatting by emulating a particular WP package that writes 119 lines of header information at the beginning of its output file. It is reading records from file SBDATA and arranging six fields from each record for output – field1 at tag1, field2 if present, field3 with preceding text all italicized, field4 after two tabs, a paragraph marker, field5 and field6 at tag2 and finally a blank line.

```
REM ----- Set tabs, tags and attributes
t1$=CHR$(9):t2$=CHR$(10)
tag1$="<newrec>":tag2$="<descript>"
italon$=CHR$(192)+CHR$(195):italoff$=CHR$(192)+CHR$(196)
REM ----- Copy header across
OPEN "TESTWP.XX" FOR INPUT
OPEN "DTPOUT.XX" FOR OUTPUT
FOR i%=1 TO 119
  INPUT LINE x$
  ? x$;
NEXT i%
```



```
CLOSE INPUT
REM ----- Now process file SBDATA
OPEN FILE"SBDATA"
SELECT FIRST FILE"SBDATA" INDEX key1.SBDATA

WHILE NOT EOF("fred")
  REM ----- first tag and field1
  ? tag1$;TRIM$(field1.SBDATA)+" ";
  REM ----- test if field2
  l%=len(TRIM$(field2.SBDATA)
  IF l% THEN ? TRIM$(field2.SBDATA) + " ";
  REM ----- italicise text and field3 (Note no trailing space)
  ? italon$;"Dated: "+DATE$(field3.SBDATA,"0d/mm/yy");italoff$;
  REM ----- two tabs field4 and a line-feed
  ? t1$t1$,TRIM$(field4.SBDATA);t2$;
  REM ----- second tag and field5 and field6 and linefeed
  ? tag2$;? TRIM$(field5.SBDATA)+" "; ? TRIM$(field6.SBDATA) + t2$
  SELECT NEXT FILE "SBDATA" INDEX key1.SBDATA
WEND
CLOSE OUTPUT
```

On completion all that remains to do is open the DTP program, load the appropriate style-sheet and "flow" in DTPOUT.XX.

Formatting a Mail Merge File for WordPerfect™

Category: Output

Covers...

Using the DML ? print command and OPEN FOR OUTPUT or OPEN FOR APPEND to format output.

The Problem

You may want to use WordPerfect to prepare form letters rather than using the Superbase Text Editor and Superbase Merge function.

The Solution

The following program will produce a mailmerge file for WordPerfect from any Superbase file. The information used in preparing this program was obtained by inspecting a number of typical merge files produced by Word- Perfect.

In particular, it was discovered that the merge "blocks" in WordPerfect may consist of more than one field. Each field is terminated by CHR\$(10), each "block" is terminated by CHR\$(18)+CHR\$(10), and each record is terminated by CHR\$(5)+CHR\$(10).

```

      REM ----- WPLET Mailmerge for WordPerfect
      DIM c1$(25,1)
      CLOSE ALL

a:
      REM ----- Select File for data
      f$ = "":REQUEST "Open file","",11,a%,f$
      IF a% = 0 THEN END
      CLS
      OPEN FILE f$

ao:
      REM ----- Strip path from filename
      f%=INSTR(1,f$,"\"): IF f% THEN f% = RIGHT$(f$,LEN(f$)-f%):GOTO a0
      b$ = "":c$ = "":b1$ = "? ":lct% = 0:blk% = 1

      REM ----- Now get fields in blocks
      FOR i% = 1 TO 25:c1$(i%,0) = "":c1$(i%,1) = "":NEXT i%

a1:
      b$ = "Block " + STR$(blk%,"99") + " = "
```

b:

```

a$ = "" : REQUEST b$, "select another or <CANCEL> to end this block",
5, a%, a$
IF a% = 0 THEN GOTO c
b$ = b$ + a$ + ", "
REM ----- reduce b$ to ensure not too long for requester
IF LEN(b$) > 60 THEN b$ = RIGHT$(b$, 58)
lct% = lct% + 1 : c1$(lct%, 0) = a$ + "." + f$
GOTO b

```

c:

```

c1$(lct%, 1) = CHR$ (18)
c$ = c$ + LEFT$ (b$, LEN (b$) - 1) + "/"
REQUEST "Another block", "", 21, a% : IF a% THEN b$ = "" : blk% = blk% +
1 : b1$ = "? " : GOTO a1
c$ = c$ + "#": lct% = lct% + 1 : c1$(lct%, 1) = CHR$ (5)
? c$

```

```

REM ----- Process file

```

d1: REM ----- Define order

```

REQUEST "Select Index to define order", "", 7, a%, i$
IF a% THEN z$ = "INDEX " + i$ + "." + f$ : EXECUTE z$

```

d2: REM ----- Ask for filter

```

SELECT WHERE ASK
SELECT FIRST
IF EOF (f$) THEN REQUEST "No records found matching this filter", "",
1, a% : IF a% THEN GOTO d2 ELSE GOTO getout
ct% = 0

```

d3: REM ----- Get merge file and check overwrite/append

```

REQUEST "Enter Merge File Name", "<CANCEL> to end", 17, a%, mf$
IF a% = 0 THEN GOTO getout
IF NOT ( EXISTS (mf$)) THEN OPEN mf$ FOR OUTPUT : GOTO d4
REQUEST "Mergefile " + mf$ + " exists", "<OK> to Overwrite or Append,
<CANCEL> to re-select", 1, a%
IF a% = 0 THEN GOTO d3
REQUEST mf$, "<OK> to Overwrite, < CANCEL > TO APPEND", 1, a%
IF a% THEN OPEN mf$ FOR OUTPUT ELSE OPEN mf$ FOR APPEND

```

d4: REM ----- Set up "? filename.file;"

```

WHILE NOT EOF (f$)
FOR i% = 1 TO lct%
IF i% <> lct% THEN zz$ = "? " + c1$(i%, 0) + ";": EXECUTE zz$
REM If end block CTRL/R

```

```
      IF c1$(i%,1) = CHR$ (18) THEN ? CHR$ (18);  
      REM If end record CTRL/E  
      IF c1$(i%,1) = CHR$ (5) THEN ? CHR$ (5);  
      ? CHR$ (10);:REM In any event <NEWLINE>  
    NEXT i%  
    ct% = ct% + 1  
    SELECT NEXT  
  WEND  
  CLOSE OUTPUT  
  ? :? &5.0ct%," records exported to " + mf$
```

```
getout:  
  SELECT WHERE  
  END
```

Modifying a Program Produced by the Report Generator

Category: Output

Covers...

Using WHERE and ORDER in reports.

Using arrays to store intermediate report data.

The Problem

The Report Generator within the Form Designer allows for a wide range of formatting, grouping, subtotaling and other report options to be defined. The act of saving the report definition then creates a Superbase SBP report program that can be modified and extended in various ways.

The Solution

This section looks at a standard report produced by the Report Generator to see how it can be modified to obtain additional results. Bear in mind that after modifying a report program it is a good idea to save it under a different name. This is because changes made to the report's visual format using the Report Generator will result in the report program of the same name being overwritten. Any program changes would then be lost.

In the example the Report Generator has been used to create a report program STKRR2.SBP. This program reports on the TRANS file contained in the Trading System application covered in the last chapter. The first few lines of output as follows:

Stock Trading Balances at Sep 11, 89

Ref: STKRR2

Stock Key

Date	b/s	Ref	Quantity	US\$ Value	Commission	Total Due
-----	---	-----	-----	-----	-----	-----
London Charter						
May 05, 89 s		000033	1200	-557.43	8.36	-549.07
May 05, 89 s		000031	2000	-929.05	13.94	-915.11
May 05, 89 s		000030	1000	-464.53	6.97	-457.56
May 03, 89 b		000025	900	425.68	6.39	432.07
May 03, 89 b		000024	1150	543.92	8.16	552.08
Apr 02, 89 b		000008	2400	1135.14	17.03	1152.17
Apr 02, 89 b		000007	2000	945.95	14.19	960.14
Apr 02, 89 b		000005	2000	844.59	12.67	857.26
Totals for London Charter				1944.27	87.71	2031.98

...and the last few lines of the report look like this:

Vancouver BBS					
May 05, 89 b	000035	1000	3463.48	51.95	3515.43
			-----	-----	-----
Totals for Vancouver BBS			3463.48	51.95	3515.43
			-----	-----	-----
			-----	-----	-----
Totals for Report		99617.432436.91		102054.34	
			-----	-----	-----

The report program STKRR2 as it was generated by the Form Designer is as follows:

OPEN FILE "DH0:SB4\STOCKS\TRANS"

```
REQUEST "REPORT TO PRINTER?", "", 134, a%
IF a% = 0 THEN END ELSE IF a% = 1 THEN PRINT ;
REPORT USD Value.TRANS, Comm_Value.TRANS, Total_Due.TRANS
AFTER REPORT
? @39; "-----"
? @1; "Totals for Report"; @40; &12 SUM USD Value.TRANS; &11 SUM
Comm_Value.TRANS; @65; &12 SUM Total_Due.TRANS
? @39; "-----"
END REPORT
```

HEADING

```
? @1; BF ; "Stock Trading Balances at "; @33; &15 DATE$ ( TODAY
, "mmm dd,yy"); BF OFF ; @65; "Ref: STKRR2"
?
? @1; "Stock Key"
? @1; "Date b/s Ref Quantity US$ Value Commission Total Due"
? @1; "-----"
END HEADING
```

```
GROUP Stock_Key.TRANS, USD Value.TRANS, Comm_Value.TRANS,
Total_Due.TRANS
BEFORE GROUP Stock_Key.TRANS
?
? @1; &24 Stock_Key.TRANS
END GROUP
```

AFTER GROUP Stock_Key.TRANS

```
? @39; "-----"
? @1; "Totals for "; @13; &15 GROUP ; @40; &12 SUM USD
Value.TRANS; &11 SUM Comm_Value.TRANS; @65; &12 SUM
Total_Due.TRANS
```

```
? @39;" -----"
END GROUP
```

```
SELECT @1; &10Trans_Date.TRANS; @12; &1Trans_Type.TRANS;
@16; &7Transactionref.TRANS; @24; &9Quantity.TRANS; @41;
&10USD Value.TRANS; @52;&11Comm_Value.TRANS; @65;
&11Total_Due.TRANS
ORDER Stock_Key.TRANS ASK
```

Fitting into the Application Environment

In the Trading System application, all programs and files are held in the same directory, but the intention is that this directory can be copied to another system or disk drive, or even called by another name.

The open file pathname accordingly requires simplifying as follows:

```
OPEN FILE "TRANS"
```

In the Trading application, all report programs are chained to by STKM, and return to STKM on completion with a flag ret% set. The flag avoids STKM having to re-specify the pull-down menus each time. A last statement is therefore added to the program as follows:

```
ret%: CHAIN"STKM"
```

Changing the Report Sequence

In the output example above, the transactions are being retrieved via the Stock_Key index to the file TRANS. With an index of this type containing duplicate keys, the first key retrieved corresponds to the last record entered. The transactions are therefore being reported in inverse chronological order. You can change this simply by adding a further condition in the DML ORDER statement to sequence on Transaction ref.TRANS.

The report program as generated includes an ASK at the end of the ORDER statement. If this is unnecessary to the application, it can simply be removed. The ORDER statement becomes:

```
ORDER Stock_Key.TRANS, Transaction ref.TRANS
```

This change puts the transactions into reference sequence as follows:

Printing a Summary Table

Now, you might want to produce a summary table at the foot of the report showing for each stock the commission earned as a percentage of the total commission earned. This tells you which stocks are generating the most commission. One way of doing this is to dimension arrays `skt$` and `skt%` to hold all the stock keys and the commissions earned. This is a realistic proposition if the number of stocks is not too large. Remember also that array dimensions are limited only by available memory. You can then insert stock keys and commission amounts into their respective arrays in the GROUP section of the program, and calculate percentages and output the summary table as part of the AFTER REPORT section.

The summary table prints as follows:

Commissions by Stock	Commission	% of total
London Charter	87.71	3.60%
London Copse	41.43	1.70%
London Corlett	97.55	4.00%
London Loden	10.82	0.44%
London UKAA	194.34	7.97%
NASDAQ EpsDes	761.82	31.26%
Sydney Stillwater	10.13	0.42%
Tokyo Cousin	289.33	11.87%
Tokyo Kobita	891.83	36.60%
Vancouver BBS	51.95	2.13%
	2436.91	100.00%

The program changes required to achieve printing of this table are as follows:

```

...
DIM skt%(40),skt$(40):sktr% = 1
...
...
tktr% = SUM Comm_Value.TRANS: EJECT :?
? @1; BF ;"Commissions by Stock "; BF OFF ; @53;"Commission % of
total"
? @53;"-----"
FOR i% = 1 TO sktr% - 1
    ? skt$(i%); @52;&11skt%(i%); @67; STR$ (skt%(i%) * 100 /tktr%,
    "999.00%")
NEXT i%
? @53;"-----"
? @52;&11tktr%; @67; STR$ (100,"999.00%")
? @53;"-----"
END REPORT

```

```
...  
...
```

```
skt%(sktr%) = SUM comm_value.TRANS:skt$(sktr%) = GROUP  
sktr% = sktr% + 1  
END GROUP
```

```
...
```

As a further refinement, the headings specified between the **HEADING** and **END HEADING** statements in the original program could be removed on the final summary table page. This is achieved by putting them into string variables at the beginning of the program, printing these string variables in the **HEADING** section, then clearing the string variables prior to the **EJECT** statement above.

Producing Statistics for Text Documents

Category: Output

Covers...

Using OPEN FOR INPUT and INPUT LINE to read a file.

Using REQUEST 13.

The Problem

When working regularly with text documents, it is often of interest to know exactly how many words, lines, sentences, paragraphs or pages there are in a document.

The Solution

This program can produce a range of statistics about any ASCII text file. The program firstly prompts you for a text file and then opens it. REQUEST 13 returns the complete DOS path to the file, as well as its name and extension. The text file is then scanned one line at a time. Lines beginning with an "Escape" character (27) are skipped, since they only contain Superbase margin settings.

We have assumed that each paragraph is separated from its predecessor by a blank line. A check ensures that multiple blank lines are not counted as multiple paragraphs. It would be possible to check for paragraphs started by indents by looking for a tab or the required number of spaces at the start of a line.

Sentences are counted by scanning each line using the INSTR function for a "." The program could be improved to search for other sentence terminators (! or ? for example). Words are defined to be sequences of characters, starting with an alphabetic character and ending with a space or an end of line. If the first character of a string is not alphabetic, it is dropped and the next character checked to be alphabetic, and so on.

Finally, a page is assumed to be 60 lines long. The number of pages is determined by dividing the total number of lines by 60.

Note: These statistics are valid for text as displayed in the text editor, or as printed using a non-proportional 10 point font. If you print the document using a proportional font, Superbase will dynamically reformat the text to fit within your margin settings, and as a result the number of lines and pages will change.

```
REM ideas92
SET PAGING OFF
REQUEST "Please select a text file for input", "", 13, req%, req$
IF req% = 0 OR req$ = "" THEN END
OPEN req$ FOR INPUT
lines% = 0: sents% = 0: words% = 0
paras% = 0: pages% = 0: isblank% = 0
```

```

WHILE NOT EOF ("")
  INPUT LINE ln$
  IF ( ASC ( LEFT$ (ln$,1)) = 27) THEN GOTO endloop
  ln$ = TRIM$ ( LTRIM$ (ln$)) + " "
  lines% = lines% + 1
  IF LEN (ln$) = 1 THEN
    IF isblank% = 0 THEN paras% = paras% + 1
    isblank% = 1
  ELSE
    isblank% = 0
    ptr% = 1:old% = 1
    WHILE ptr% <> 0
      ptr% = INSTR (old%,ln$,".")
      IF ptr% > old% THEN sents% = sents% + 1
      old% = ptr% + 1
    WEND
    ptr% = 1:old% = 1
    WHILE ptr% <> 0
      ptr% = INSTR (old%,ln$," ")
      IF ptr% > old% THEN
        words% = UCASE$ ( MID$ (ln$,old%,ptr% - old%))
        WHILE LEN (word$) > 0
          st1$ = ASC ( LEFT$ (word$,1))
          IF st1$ > 64 AND st1$ < 91 THEN
            words% = words% + 1
            word$ = ""
          ELSE
            IF LEN (word$) THEN word$ = RIGHT$ (word$) - 1)
          END IF
        WEND
      END IF
      old% = ptr% + 1
    WEND
  END IF

endloop:
WEND
CLOSE INPUT
pages% = INT (lines% / 60)
IF lines% MOD 60 <> 0 THEN pages% = pages% + 1
CLS
? UL "Document statistics for file: ";req$ ATTR OFF
?
? "Total Words    :"; STR$ (words%,"999999")
? "Total Sentences :"; STR$ (sents%,"999999")

```

```
? "Total Lines    :"; STR$ (lines%, "999999")
? "Total Paragraphs :"; STR$ (paras%, "999999")
? "Total pages    :"; STR$ (pages%, "999999")
SET PAGING ON
END
```

Producing a Word List and Count for a Text File

Category: Output

Covers...

Using CREATE, ADD and MAKE to create a temporary Superbase file.

The Problem

You may want to process a text file with a view to producing a sorted output list, say of words in the file together with the number of occurrences. Using dimensioned arrays is inappropriate, since the number of elements is indeterminate and may be very large.

The Solution

This problem and many list processing problems like it can be handled by creating a temporary Superbase file.

In creating a Superbase file under DML, the normal rules for Superbase files must be obeyed; the file must have at least one index (denoted by IXD or IXU in the file definition). The full range of Superbase field types is available, as are calculations, validations and other field attributes.

The following example program takes an input text file, extracts from it a list of all the words used and the number of times each word occurs, and displays the results in alphabetical order. Words in this instance are defined to be sequences of characters, starting with an alphabetic character and ending with a space or an end of line. If the first character of a string is not alphabetic, it is dropped and the next character checked to be alphabetic, and so on.

As it stands the program is incomplete, as no checking is done for punctuation or other marks at the ends of words. It is included here as an example of the use of temporary files:

```
REM ideas93
SET PAGING OFF
IF EXISTS ("TempFile.SBF") THEN
  OPEN FILE "TempFile"
  REMOVE FILE "TempFile"
END IF
CREATE "TempFile"
ADD "Word;TXT IXD;40;1,1"
ADD "Instances;NUM;9999.;2,1"
MAKE "TempFile"
REQUEST "Please select a text file for input", "", 13, req%, req$
IF req% = 0 OR req$ = "" THEN END
OPEN req$ FOR INPUT
```

```

WHILE NOT EOF ("")
  INPUT LINE x$
  x$ = LTRIM$ (x$)
  IF x$ = "" THEN GOTO endl ine
  IF ASC (RIGHT$ (x$,1)) < 33 THEN x$ = LEFT$ (x$,LEN (x$) - 1)
  x$ = x$ + " "
  ? x$
  front% = 1:back% = 1
  WHILE back% <> 0
    back% = INSTR (front%,x$," ")
    IF back% > front% THEN
      word$ = LTRIM$ ( MID$ (x$,front%,(back% - front%)))
      word$ = UCASE$ (word$)
      IF LEN (word$) > 40 THEN word$ = LEFT$ (word$,40)
      WHILE LEN (word$) >0
        a% = ASC ( LEFT$ (word$,1))
        IF a% > 64 AND a% < 91 THEN
          IF EXISTS (word$,Word.TempFile) THEN
            SELECT KEY word$
            Instances = Instances + 1
            STORE FILE "TempFile"
          ELSE
            BLANK FILE "TempFile"
            Word = Word$
            Instances = 1
            STORE FILE "TempFile"
            uniq% = uniq% + 1
          END IF
          words% = words% + 1:word$ = ""
        END IF
        IF LEN (word$) THEN word$ = RIGHT$ (word$, LEN (word$) - 1)
      WEND
      front% = back% + 1
    END IF
    IF back% = front% THEN front% = front% + 1
  WEND

endl ine:
WEND
CLOSE INPUT
CLS: SET TABLE: SET PAGING ON
? "Alphabetic listing of all words used"
SELECT FIRST :VIEW
WHILE NOT EOF ("TempFile")
  SELECT NEXT :VIEW

```

WEND

? "Number of words = "; STR\$ (words%, "999999")

? "Number of unique words = "; STR\$ (uniq%, "999999")

?

INDEX

A

Adding Password Protection 23-3
Application help 21-29
Applications
General ledger 23-9
Inventory control 23-5
Mailing list processing 21-1 - 21-31
Mailmerge with Word Perfect 23-16
Online conferencing 23-7
Producing statistics for text files 23-25
Producing word lists 23-28
Share pricing 23-7
Share trading 22-1 - 22-30
Arrays 23-19
ASCII disk files 23-12
ASCII text files 23-25, 23-28
Autodialing 23-6

B

Backing up 21-20
Browsing 22-5, 22-40
Browsing errors 22-40

C

Carriage returns 23-9, 23-12
Check boxes 22-34
Clerical procedures 22-3
Color conventions 22-34
Command strategies 21-7
Communications 23-5, 23-7
Customizing the Superbase Environment 23-1 - 23-30

D

Daily activity cycle 22-6, 22-27
Database files
CASH 22-19, 22-49
CERTS 22-53
CLIENTS 22-8, 22-56
COUNTRY 22-11, 22-41

CTRL 22-20
CURRENCY 22-11, 22-14, 22-42,
22-47
DEAGL 23-10
HELPBASE 21-3, 21-29
MAILBASE 21-2
MDEFAULTS 21-12
RULEBASE 21-3
STOCKS 22-11, 22-33, 22-47, 22-56
TRANS 22-15, 22-34, 22-47, 23-19
Database structure 22-4
Date and time display 22-26
Default values in fields 22-16
Deleting data 22-39, 22-45
Desktop publishing
See DTP
DML
? 23-13, 23-16
? QUERY 22-28
? TEXT MERGE 22-55
?MEMORY 21-13
ADD 23-28
BLANK FILE 23-10
BLANK FORM 21-8, 22-37
BREAK OFF 23-1, 23-7
CHAIN 22-29, 23-2, 23-21
CLOSE FIELDS 21-19
CLOSE FORM 22-27
CLOSE INPUT 21-14, 21-21, 23-14
CLOSE OUTPUT 21-13, 23-15
Comms commands 23-5, 23-7
CREATE 23-28
CREATE INDEX 21-16, 21-22
DATA 21-30
DIRECTORY 21-6, 23-1
EDIT 22-30
EDIT QUERY 22-28
END 22-27
ENTER 21-8, 22-37
ENTER ROW 22-48
ERR\$ 22-30
ERRNO 22-30
EXECUTE 21-23, 21-26, 23-18
EXPORT 21-20
FN sys() 22-23

FORM 22-23
GET 21-9, 21-17, 21-29
GROUP 23-20
HEADING 23-20, 23-24
IF EOF () THEN 21-17, 21-24
IMPORT 21-14
INDEX 21-15, 21-18, 22-36
INPUT 23-9
INPUT LINE 21-14, 21-21, 23-12,
23-25
KEY 21-7 - 21-8
LABELS 21-19
LOAD LABELS 21-19
LOAD QUERY 22-28
MAKE 23-28
MENU 21-6, 22-25
MENU CLEAR 22-25
MENU ON 21-7, 22-23
MERGE TEXT 22-55, 23-2
MODIFY 21-16
NEW QUERY 22-28
NUMBASE 22-24, 22-27
ON ERROR GOTO 21-5, 22-23,
22-28, 22-40
OPEN FILE 22-24
OPEN FOR APPEND 23-16
OPEN FOR INPUT 23-9, 23-12,
23-25
OPEN FOR OUTPUT 21-13, 23-12,
23-16
OPEN FORM 21-8, 22-23 - 22-24,
22-27, 22-31
ORDER 23-21
PANEL ON/OFF 22-24, 22-27, 23-1
PRINT 22-29
PROTECT 23-3
QUIT 23-1
READ 21-14
RECCOUNT 21-23
REMOVE INDEX 21-15
REPORT 23-20
RESTORE 21-14
RESUME 22-30
SAVE QUERY 22-28
SELECT 22-2, 23-3, 23-20

SELECT ALL 21-20, 21-22
SELECT CASE 21-7, 21-11, 21-19,
22-27
SELECT CURRENT 21-17
SELECT DUPLICATE 21-17
SELECT FIRST 21-16, 22-43
SELECT FORM KEY 22-31
SELECT FORM NEXT PAGE 22-32
SELECT FORM PREVIOUS PAGE
22-32
SELECT FORM ROW 22-45
SELECT KEY 21-18, 22-46
SELECT LAST 22-43
SELECT NEXT 21-17, 22-46
SELECT PREVIOUS 22-46
SELECT REMOVE 21-15, 21-18,
22-46
SELECT WHERE 21-23, 22-40
SER 21-2, 22-10, 22-16, 23-10
SET 21-5
SET HEADING 21-8, 21-11, 22-26 -
22-27
SET PAGING ON/OFF 22-24, 23-7
SET REQUEST HEADING 22-24,
22-27
SET STATUS 21-5, 21-8, 22-23
STORE FORM 21-8
TRIM\$ 23-13
UPDATE 21-22
UPDATE WHERE 22-57
WAIT COMMS OR KEY 23-6
WAIT KEY 21-15
WAIT MENU 21-7, 22-23
WAIT MOUSE 22-37
WAIT PANEL 22-40
WAIT PANEL OR KEY 21-9
WHILE WEND 22-23
Document production 22-6, 22-53
DTP
Carriage returns 23-12
Line feeds 23-12
Space characters 23-13
Tab characters 23-13
Duplicate record checking 21-5, 21-13

E

Editing data 22-38
Entering data 21-8, 22-37, 22-45, 22-49
Error trapping 21-31
Exit routines 21-30, 22-26
Exporting 21-20

F

Field indirection 22-36, 22-43
Fields
 Calculated 22-8, 22-13, 22-16
 Constant 22-16
 Date, time 22-12, 22-15 - 22-16, 22-19
 External image 22-10
 External text 22-10
 Required 22-15
 Serial 21-2, 22-10, 22-16, 23-9
 Text 21-2, 23-13
 Validated 22-11
 Virtual 21-2, 22-10, 22-19 - 22-20
File header information 23-14
File maintenance 22-5, 22-31
Filenaming conventions 22-6
Filter 23-22
Flags 22-18
Form calculations 22-56
Form color 21-4, 22-7
Form commands 22-31 - 22-32
Form letters 22-53
Form transactions 22-41, 22-56
Formatting for DTP 23-12
Forms
 ENTRY 21-4
 MENUF 22-23
 STKC 22-8, 22-31
 STKC2 22-8
 STKDC 22-14, 22-47
 STKDN 22-18, 22-47
 STKDR 22-19, 22-49
 STKDS 22-11, 22-47
 STKO 22-11, 22-41
 STKO2 22-41
 STKOS 22-8, 22-15, 22-56
 STKOS2 22-8, 22-15
 STKS 22-11, 22-33

STKS2 22-11
STKT 22-15, 22-34
STKU 22-14, 22-42
Function key assignments 21-10

G

Grayed menu items 22-22

H

Help 21-29
Housekeeping 21-19, 22-4

I

Importing 21-13, 23-9
Non-ASCII data 23-9
Indexing
Duplicate keys 21-15
On a virtual field 22-12, 22-18
Unique keys 22-8

L

Labeling conventions 22-26
Line feeds 23-9, 23-12
LOOKUP 22-2, 22-4, 22-8, 22-11, 22-13 - 22-15, 22-17

M

Mailing labels 21-18
Mailmerge 22-53
Formatting for Word Perfect 23-16
Menu graying 22-2, 22-6
Menu options
Activating 22-18, 22-20, 22-57
Graying 22-18, 22-20, 22-57
Keyboard selection 21-6
Menus
Form-based 21-4
Pull-down 21-7, 22-5, 22-22
Screen list 23-1
Merging Mailing lists 21-1, 21-11
Modifying a file definition 21-5
Modifying report programs 23-19
Modular structure 22-26
Multi-page forms 22-31, 22-33

N

Networking 23-5
Non-ASCII fixed length files 23-9

O

One-to-many replication 22-1

P

Processing rules 22-2
Program chaining 22-29
Program control of applications 21-1, 21-5, 22-1, 22-22
Program listings 20-2
Program routines within MENU
fdelete 22-31, 22-39
fedit 22-31, 22-38
fenter 22-31, 22-37
ferror 22-30
fexit 22-31
floop 22-36
fswitch 22-32, 22-41
fwait 22-36
m1 22-27
m2 22-27
m3 22-27
m31 22-53
m4 22-27
m41 22-56
m42 22-57
m43 22-57
m44 22-58
m5 22-27
m6 22-29
mloop 22-23
sdorp 22-29, 22-55
shead 22-26
sindex 22-31, 22-39
sinit 22-24
smenu 22-25
spanel 22-31, 22-40
sprint 22-31
sssearch 22-36
sset 22-36
stkf 22-31, 22-35, 22-56
supdate 22-38
Program routines within STKFO

fcurr 22-46
fdelete 22-46
fenter 22-45
fform 22-43
floop 22-43
fnext 22-46
fpage 22-43
fprev 22-46
fwait 22-44
sinteg 22-44
slimits 22-43
spage 22-43
Programs
IDEAS92 23-25
IDEAS93 23-28
MAILBASE 21-1
MAILIT 21-5
MENU 22-1, 22-8, 22-15, 22-22 - 22-23
START 22-1, 23-1, 23-3
STKDC 22-11, 22-14, 22-47 - 22-48
STKDR 22-15, 22-19, 22-49
STKFO 22-11, 22-14 - 22-15, 22-41 - 22-42
STKFU 22-34, 22-42
STKRR2 23-20
WPLET 23-16
Purging Mailing lists 21-1, 21-11
Pushbuttons 21-4, 22-5, 22-31 - 22-32, 22-41 - 22-42

Q

Query 22-6
Querying under program control 22-27

R

Radio buttons 21-4, 22-34, 22-49
Record numbers 22-8
Referential integrity 22-44
Report generator 22-29, 23-19
Reporting 22-6, 23-19
Printing summaries 23-22
Reports
STKRB 22-19
STKRR 22-19
STKRR2 23-20
Request
4 21-27

5 23-17
9 21-26
11 21-20 - 21-21, 23-16
13 23-25
17 23-18
18 21-18
23 23-1, 23-4
113 21-21
130 21-14, 21-28
143 21-21
REQUEST type
100 22-44
114 22-30
119 22-45
130 22-30, 22-55
14 22-28
140 22-53, 22-56
2 22-30
20 22-9, 22-12, 22-15
4 22-16, 22-29
7 22-39
Restoring 21-21
Runtime system 20-2, 22-1, 22-22

S

Sending and receiving text 23-7
START 22-1, 23-1, 23-3
Superbase 22-1, 22-22

T

Temporary files 22-53, 23-28
Ternary calculations 22-13, 22-17
Text utilities 23-25, 23-28
Time and date display 22-23

U

Unattended File Transfer 23-5
Updating 21-22, 22-6, 22-18, 22-56

W

Word count 23-28
Word list 23-28
Word Perfect 23-16
Work flow 22-4

—

—

—

